Osa-kokonaisuussuhteen käsittelyyn kykenevän helppokäyttöisen kyselykielen suunnittelu ja toteutus

Samu Viita

Tampereen yliopisto
Tietojenkäsittelytieteiden laitos
Pro gradu -tutkielma
Maaliskuu 2002

Tietojenkäsittelytieteiden laitos
Viita Samu
Pro gradu -tutkielma, 74 sivua
Maaliskuu 2002

Tiivistelmä
Tutkielmassa toteutetaan osa-kokonaisuussuhteiden käsittelyyn soveltuvan kyselykielen prototyyppi. Myös kielen ominaisuuksia ja sen pohjalla olevaa mallinnusmenetelmää esitellään. Suunnittelussa on kiinnitetty ensisijaisesti huomiota kyselykielen helppokäyttöisyyteen ja osa-kokonaisuussuhteen erityispiirteiden huomioon ottamiseen. Erityispiirteitä tukee muun muassa intensionaalisten tietojen tarkastelun mahdollistaminen kyselyissä.

Usein olemassaolevissa kyselykielissä osa-kokonaisuushierarkioiden käsittely on monimutkaista ja kyselyjen ilmaisu muistuttaa ohjelmointia. Kehitetyn kielen käyttäjältä ei sen sijaan vaadita ohjelmointitaitoja. Kielen deklaratiivisuuden asteen on tarkoitus olla mahdollisimman korkea. Kielen käyttäjän ei tarvitse hallita rekursiota, iteraatiota, mallinsovitusta tai muita ohjelmointitekniikoita. Kielessä ei tarvitse navigoida osa-kokonaisuusrakenteessa polkuesityksellä. Osa-kokonaisuushierarkiaa ei myöskään tarvitse purkaa hierarkiaa sisältämättömäksi rakenteeksi. Sekä navigointi, että hierarkiarakenteen purku hierarkittomaksi vaativat tarkkaa tietoa intensionaalisesta-, eli kaaviotasosta. Navigointi on yleistä oliotietokannoissa. Rakenteen purkaminen hierarkittomaksi on puolestaan tyypillistä nf^2-mallille.

Tutkielmassa kehiteltävä kyselykielen toteutus pohjautuu deduktiiviseen olio-orientoituneeseen tietokantaparadigmaan. Osa-kokonaisuussuhteen mallinnus perustuu indeksointimekanismiin. Sen avulla intensionaalinen ja ekstensionaalinen taso saadaan sidottua toisiinsa kaksisuuntaisesti. Lisäksi mekanismilla pystytään analysoimaan osa-kokonaisuusrakennetta monipuolisesti. Mallinnus toteutettiin Prolog++ -ohjelmointikielellä, joka tuki kielen toteuttamista hyvin.

Avainsanat ja -sanonnat: osa-kokonaisuussuhde, deduktiivinen oliotietokanta, kyselykieli, indeksointimekanismi.

Termit

C++	Proseduraalinen ohjelmointikieli, jossa on olio-ohjelmointikielen ominaisuuksia.
Deklaratiivinen ohjelmointikieli	Ohjelmointikieli, jolla lopputulos kuvaillaan korkealla abstraktiotasolla yksityiskohdista välittämättä
Hybridikieli	Ohjelmointikieli, joka yhdistää useampia ohjelmointiparadigmoja
Proseduraalinen ohjelmointikieli	Ohjelmointikieli, jossa ohjelmoija määrää lopputuloksen vaihe vaiheelta ja rivi riviltä. Vastakohta deklaratiiviselle kielelle
SQL	Structured query language. Yleinen relaatiotietokantojen yhteydessä käytetty deklaratiivinen kyselykieli.

1. Johdanto

Osa-kokonaisuussuhde (part-of-, part-whole-, whole-part -relationship) on tärkeä käsitteellinen mallintamisrakenne tietokantojen yhteydessä. Sitä käytetään rakenteellisen informaation, kuten esimerkiksi rakenteellisesti monimutkaisten olioiden mallintamiseen. Tällaisen mallintamisen avulla voidaan kuvata monenlaisia abstrakteja tai konkreettisia asioita. Osa-kokonaisuussuhteen mallintamista ja käytetään monilla sovellusalueilla, maantieteellisissä informaatiosysteemeissä (GIS) [Price *et al*., 2000], dokumenttien hallinnassa [Lambrix, 2000], lääketieteellisissä sovelluksissa [Hanh *et al.*, 1999] ja tekoälyn sovelluksissa [Lambrix, 2000]. Kokonaisuuksia voidaan käyttää myös tietokannoissa lukituksen, tiedonvarastoinnin, tietokantahaun ja versioinnin yksikkönä. [Kim *et al*., 1987a; Kim *et al*., 1987b].

Osa-kokonaisuussuhteet perustuvat kahteen keskeiseen käsitteeseen, *komposiittiin* ja *komponenttiin*. Komposiitti tarkoittaa kokonaisuutta, joka muodostuu komponenteista. Komponentti on puolestaan osa jotain komposiittia. [Wand *et al.*, 1999] Komponentti ja komposiitti ovat keskinäisessä vuorovaikutuksessa monella tavalla. Komposiitilla on usein ominaisuuksia, jotka perustuvat sen osien ominaisuuksiin tai esimerkiksi sen kaikkien osien yhteisvaikutukseen. Lisäksi komponentilla saattaa olla ominaisuuksia, jotka riippuvat sen komposiitin ominaisuuksista. Komposiitin ja komponentin olemassaolo voi myös riippua toistensa olemassaolosta.

Osa-kokonaisuussuhde ei välttämättä rajoitu pelkästään välittömiin komponentteihin tai komposiitteihin. Rakenteellisesti monimutkaisilla objekteilla saattaa olla myös välillisiä komponentteja. Välillistä komponenttia voidaan havainnollistaa seuraavalla esimerkillä. Oletetaan, että kompositiolla A on komponenttina B. B:llä on edelleen komponenttina C ja C:llä komponenttina D. Tällöin A:n välitön komponentti on B ja sen välillisiä komponentteja ovat C ja D. Sekä välittömät, että välilliset komponentit tulkitaan usein kokonaisuuden komponenteiksi. Myös välilliset komposiitit tulkitaan samalla tavalla, mutta käänteisesti. Tällaisia välillisiä suhteita sanotaan myös *transitiivisiksi* suhteiksi. Reittiä komposiitista komponenttiin tai päinvastoin kutsutaan *poluksi.*

Tietokannoista tietoja haetaan kyselykielten avulla. Monesti olemassaolevilla kyselykielillä osa-kokonaisuussuhteen käsittelyn ilmaiseminen on vaikeaa loppukäyttäjälle. Käsittely saattaa edellyttää rekursiivisen ja iteratiivisen ajattelutavan sisäistämistä. Usein edellytetään myös tietoa esitystavan yksityiskohdista. Lisäksi on yleistä, että transitiivisia suhteita sisältävissä rakenteissa täytyy etsiä haluttu komponentti rakenteesta navigoimalla polkuesitystavalla. Toi-

nen tapa poistaa kompleksisuutta on purkaa komponenttien välillä vallitseva hierarkia. Kummassakin tapauksessa osa-kokonaisuusrakenne täytyy entuudestaan tuttu, jotta käsittely onnistuu.

Osa-kokonaisuussuhteessa tieto esitetään ja käsitellään sekä *intensionaalisella* että *ekstensionaalisella* tasolla. Intensionaalinen taso tarkoittaa kaaviotasoa, jossa määritellään eksplisiittisesti komponentti- ja komposiittityypit, tyyppien ominaisuudet sekä niiden keskinäiset suhteet. Ekstensionaalisella tasolla on intensionaalisella tasolla määriteltyjen käsitteiden ilmentymät. Jos jostakin intensionaalisella tasolla määritellystä käsitteestä muodostetaan ilmentymä, sen ominaisuudet, eli attribuutit arvottuvat.

Pääosin useimmat kyselykielet on tarkoitettu ekstensionaalisten tietojen hakuun esittämällä intensionaalisia kriteereitä. Intensionaalisen tason monipuolista tarkastelua ei kuitenkaan usein tueta. Intensionaalisten asioiden tarkastelu lisää kuitenkin ilmaisuvoimaa etenkin osa-kokonaisuussuhteita käsitteleville kyselykielille. Tässä tutkielmassa kehitetyssä kielessä molempien tasojen monipuolinen tarkastelu on mahdollista.

Tutkielmassa tarkastelen osa-kokonaisuussuhteen mallintamisratkaisua ja kehitän siihen perustuvaa kyselykieltä. Mallintamisratkaisun ja siihen perustuva kyselykielen toteutan logiikkaohjelmointi- ja olioparadigman yhdistävällä, Prolog++ -hybridikielellä. Mallintamisratkaisun periaate perustuu Timo Niemen [1983] arvo-orientoituneeseen esitystapaan, jota Niemi, Kalervo Järvelin ja Marko Junkkari ovat myöhemmin kehitelleet. Erityisen suurta huomiota kiinnitän kyselykielen korkeaan deklaratiivisuuden asteeseen käyttäjäystävällisyyden takaamiseksi. Tutkielmani päätarkoituksena on kehittää kyselykieli, joka ottaa huomioon osa-kokonaisuussuhteen luonteen vaatimukset ja erityispiirteet.

2. Osa-kokonaisuussuhteen luonnehdinta

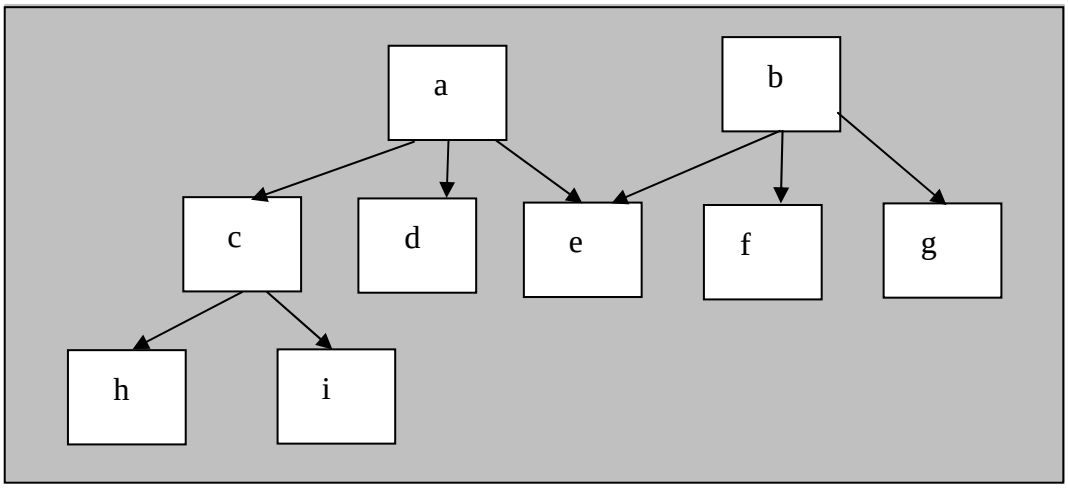
2.1. Osa-kokonaisuussuhteen yleiskäsitteitä

Osa-kokonaisuussuhde on käsitteenä monimerkityksellinen, joten sen tarkka rajaaminen on vaikeaa. Luonteenomaista suhteelle on, että se pyrkii kuvaamaan itsenäisesti tunnistettavaa kokonaisuutta, joka muodostuu useammasta kuin yhdestä tunnistettavasta osasta. Tällöin osat eivät saa olla pelkkiä ominaisuuksia, vaan tavallaan itsenäisesti tunnistettavia objekteja, joilla on omia ominaisuuksia. Lisäksi osalla on oma merkityksensä kokonaisuuden kannalta. Tämä ei kuitenkaan tarkoita, että osia ja kokonaisuuksia voisi aina erottaa toisistaan. Kokonaisuus on puolestaan osiensa muodostama, ja sillä on aina joitakin uusia ominaisuuksia, joita sen osilla ei ole. Voidaankin sanoa, että kokonaisuus on enemmän kuin osiensa summa.

Osien suhteita on tutkittu formaalilla tasolla *mereologiassa* (filosofian eräs alue) ja kokonaisuuksien teoriaa *topologiassa*. Lisäksi on olemassa *mereotopologia*, joka yrittää yhdistää kaksi aikaisemmin esitettyä teoriaa [Varzi, 1996]. Mereologiassa tutkitaan osien suhteita muihin osiin. Teoria tunnistaa transitiivisuussuhteen osien välillä. Transitiivisuusominaisuuden huomioonotto on tutkielmassa keskeisessä osassa. Mereologiassa tunnistetaan lisäksi osien välillä oleva symmetrisyys ja refleksiivisyys. Mereologinen symmetrisyys tarkoittaa, että jos osalla a on osa b ja osalla b on osa a, niin tällöin a = b. Refleksiivisyys puolestaan tarkoittaa, että osa on itsensä osa. Tässä tutkielmassa symmetrisyys pätee, mutta refleksisyys ei. Tämä johtuu siitä, että usein arkielämässä osan ei katsota olevan itsensä osa. Tutkielman kyselykieli on tarkoitettu mahdollisimman helppokäyttöiseksi ja refleksiivisyysominaisuuden noudattaminen saattaisi aiheuttaa väärinkäsityksiä.

Mereologia keskittyy ainoastaan ekstensionaaliseen tasoon. Tämä seikka aiheuttaa ongelmia erityisesti olemassaolon periaatteen kohdalla. Periaatteen mukaan kaksi objektia ovat identtisiä keskenään, jos niillä on samat osat samalla ajan hetkellä. Tämä vaatimus on liian tiukka käytännön tilanteisiin [Lambrix and Padgham, 2000], mukaan lukien kyselykielet. On esimerkiksi mahdollista olla kaksi eri tahon julkaisemaa konferenssijulkaisua, joissa on kuitenkin samat artikkelit.

Seuraavaksi esitän tutkielman kyselykielen kannalta osa-kokonaisuussuhteessa olevia olennaisia käsitteitä. Kuva 1. demonstroi osa-kokonaisuussuhteen eri käsitteitä graafisesti.



Kuva 1. Esimerkki osa-kokonaisuussuhteesta

Perusolioksi (basic object) kutsutaan oliota, jolla ei ole komponentteja. Kuvassa 1. perusolioina ovat d, e, f, g, h ja i. *Ylimmän tason olio (super object)* puolestaan tarkoittaa oliota, jolla ei ole kompositiota. Näitä ovat kuvassa a ja b. *Polku (path)* jostain oliosta toiseen osa-kokonaisuussuhteessa olevaan olioon tarkoittaa reittiä, joka yhdistää oliot a ja b toisiinsa. Esimerkissä *Polku* oliosta a olioon i koostuu reitistä a-c-i. *Jaetuksi osaksi (shared object)* kutsutaan osaa, joka on vähintään kahden komposiitin komponenttina. Kuvassa 1 olio e on tällainen.

Osa-kokonaisuussuhteen mallintamisen tarkoitus on kohdealueen kokonaisuuden ja sen osien välisen suhteen esittäminen. Kirjallisuudessa puhutaan usein myös kompleksisista objekteista [Järvelin and Niemi, 1999; Savnik *et al*. 1999; Carey *et al*., 1988] ja komposiittiobjekteista [Halper *et al*., 1998; Motscnig-Pitrik and Kaasboll, 1999]. Vaikka monet puhuvat aggregaatiosuhteista osa-kokonaisuussuhteiden yhteydessä, termi on moniselitteinen.

Motschnig-Pitrik ja Kaasboll [1999] tekee erottelun komposition ja aggregaatin välille. Aggregaatin ja komposition semantiikat eroavat toisistaan heidän erottelussaan seuraavalla tavalla: aggregaatti voi olla jokin kompleksinen rakenne, jossa jäsenillä ei ole komponentti- tai komposiittisuhdetta keskenään. Esimerkiksi tästä Motschnig-Pitrik kollegoineen ottaa huoneen, joka voidaan tulkita joko aggregaatiksi tai komposiitiksi. Jos huone tulkitaan koostuvan lattiasta, seinistä, ikkunoista ja ovista, puhutaan kompositiosta. Jos taas huone nähdään siihen liittyvien ominaisuuksien joukkona, kuten esimerkiksi omistaja, koko ja sijainti, tulkitaan huone aggregaatiksi. Aggregaatti on siis erottelussa käytännöllisemmässä ja yleisemmässä merkityksessä kuin komposiitti. Aggregaatin yhteydessä huoneeseen yhdistetään attribuutteja, joilla on merkitys jossakin sovelluksessa. Tässä tapauksessa attribuuttien tarkoituksena ei ole kui-

tenkaan olla itsenäisiä objekteja, vaan mallintaa huoneeseen liittyviä ominaisuuksia. Huoneen osina ei siis ole omistaja eikä sijainti, vaikka ne mallinnusrakenteen kannalta liitetäänkin huoneeseen. Ominaisuudet voivat kuitenkin aggregaatissa olla kompleksisia rakenteita. Huone -esimerkissä muun muassa huoneen omistajalla voi olla attribuutteja ja komponentteja. Sen sijaan vastaava kompositio -esimerkki oli selvästi tarkoitettu mallintamaan osa-kokonaisuussuhdetta. Tässä työssä pidättäydyn samassa erottelussa, jotta väärinkäsityksiltä vältyttäisiin.

Transitiivisuus on keskeisessä roolissa osa-kokonaisuussuhteissa. Esimerkkinä tarkastelen autoa, joka edustaa kokonaisuutta. Auton renkaan pölykapseli on renkaan komponentti ja rengas puolestaan auton komponentti. Toisin sanoen pölykapseli on myös auton komponentti. Komposiitin kohdalla tulkinta on käänteinen: koska auto on renkaan komposiitti ja rengas pölykapselin komposiitti, silloin myös auto on pölykapselin kompositti. Transitiivinen tulkinta ei kuitenkaan ole itsestäänselvyys kaikissa tilanteissa [Winston *et al.,* 1987; Iris *et al.*, 1988; Artale *et al.*, 1996]. Mielekkyys katoaa muun muassa seuraavassa esimerkissä: käsi on jalkapalloilijan osa ja jalkapalloilija on jalkapallojoukkueen osa, mutta kättä ei ole mielekästä enää tulkita joukkueen osaksi. Transitiivinen tulkinta ei ole edellä mielekäs, koska käden ja ihmisen suhteen luonne on erilainen, kuin ihmisen ja joukkueen suhteen luonne. Onkin huomattu, että transitiivinen tulkinta saattaa menettää mielekkyytensä, jos luonteeltaan erilaisia suhdetyyppejä sekoitetaan mielivaltaisesti keskenään [Winston, 1987]. Tämän takia on syytä erottaa ja tunnistaa erilaisia osa-kokonaisuussuhdetyyppejä.

2.2. Osa-kokonaisuussuhdetyypien jaotteluja

Osa-kokonaisuussuhteiden eri tyyppien jaottelu on vaikea tehtävä. Kaikki suhdetyypit eivät ole selkeästi luokiteltavissa johonkin tiettyyn tyyppiin kuuluvaksi. Monissa jaotteluyrityksissä on inhimillisellä tulkinnalla merkitystä, eikä tulkintaa voida poistaa tässäkään tutkielmassa. Lisäksi eri näkökulmat painottuvat tieteenalasta riippuen, jossa suhteita tutkitaan. Pääasia kuitenkin tutkielman kyselykielen kannalta on se, että käytettävässä sovellusalueessa transitiivisuus pätee osien ja kokonaisuuksien kesken.

Kielitieteen ja kognitiotieteen saralla on tehty ontologinen erotteluehdotus [Winston *et al.*, 1987], joka pohjautuu erilaisiin osa-kokonaisuussuhteiden tyyppeihin. Seuraavaksi esittelen yhteenvedon tästä erotteluehdotuksesta:

Komponentti/kokonaisuuden muodostava olio: Kokonaisuuden muodostavilla olioilla on rakenne, jonka muodostavat erilliset komponentit, joilla on tietty oma merkitys tai toiminta kokonaisuuden kannalta. Esimerkiksi renkaat

ovat osa autoa ja fonologia on osa kielitiedettä. Tämä tyyppiset osa-kokonaisuussuhteet ovat pääosin niitä, johon tutkielman kyselykieli keskittyy.

Jäsen/kokoelma: Muodostaa käsityksen jäsenyydestä suhteessa tiettyyn olioiden kokoelmaan. Jäsenillä ei ole toiminnallista roolia kokonaisuudessa, mutta ne voidaan tunnistaa kokonaisuudesta. Esimerkiksi puu (jäsen) on osa metsää (kokoelma).

Osa, osuus massasta: Kokonaisuutta tarkastellaan homogeenisenä koosteena. Lisäksi sen osuudet tai osat ovat samankaltaisia kuin kokonaisuus, mutta kuitenkin erotettavissa siitä. Esimerkiksi siivu on osa piirasta.

Aine/olio: Tyyppi ilmaisee aineksena olevan olion: ”… on osittain puuta” tai ”… on tehty jostakin”. Ainetta, josta olio on tehty, ei voida erottaa oliosta eikä sillä ole funktionaalista roolia. Esimerkiksi lentokone on tehty osaksi alumiinista.

Vaihe/toiminta: Tämä typpi ilmaisee aktiviteetin vaiheen. Vaihe on kuin komponentti, jolla on toiminnallinen rooli, mutta se ei ole kuitenkaan erotettavissa irralleen koko toiminnasta. Esimerkiksi halkoon tarttuminen on osa halkojen pinoamista.

Paikka/alue: Avaruudellinen relaatio alueiden kesken, joilla eri oliot sijaitsevat. Osat kuuluvat johonkin suurempaan alueeseen, mutta niitä ei voi erottaa tästä alueesta erilleen. Esimerkiksi lähde on osa aavikkoa.

Edellä esitetyssä jalkapalloesimerkissä transitiivisuusominaisuuden mielekkyys kadotettiin, koska siinä sekoitettiin komponentti/kokonaisuus-suhde ja jäsen/kokoelmasuhde keskenään.

Yllä esitetty jaottelu on saanut myös kritiikkiä siitä, että sen näkökulma on liian kielitieteellinen. Lisäksi jotkin suhdetyypit ovat liian tulkinnanvaraisia. Tämän vuoksi jotkin erityispiirteet ovat hämäriä ja kiistanalaisia. Seuraavassa kahdessa erotteluehdotuksessa on yritetty vähentää tulkinnanvaraisuutta.

Ensiksi tarkasteltavassa jaotteluehdotuksessa tunnistetaan kuuden suhdetyypin sijasta neljä suhdetyyppiä [Iris *et al.*, 1988]. Ne ovat funktionaalinen komponenttisuhde, lohko/kokonaisuussuhde, jäsen/kokoelma -suhde ja osajoukko/joukko -suhde. Funktionaalisessa komponenttisuhteessa komponentti ei ole ainoastaan rakenteellinen yksikkö, vaan osalla on myös tärkeä tehtävä tai toiminta kompositiossa. Esimerkkinä tästä on polkupyörä ja rengas, jolloin renkaalla on tärkeä tehtävä polkupyörän toiminnassa. Lohko/kokonaisuussuhteen tarkoitus on kuvata tilannetta, jossa kokonaisuudesta on eroteltavissa tai siirreltävissä osia. Osat ovat kuitenkin samankaltaisia, kuin kokonaisuus ja kokonaisuuden täytyy olla olemassa ennen kuin osat ovat olemassa. Kaksi viimeistä erottelua, jäsen/kokoelma ja osajoukko/joukko ovat samat kuin Winstonin ja kumppaneiden [1987] erottelussa.

Toisessa jaottelussa [Gerstl and Pribbenow, 1995] osa-kokonaisuussuhteet erotellaan komposiittien rakenteen mukaan. Eri suhdetyyppejä on kolme: *Komplekseilla* on heterogeeninen rakenne ja niiden osia kutsutaan komponenteiksi. Osien ja kokonaisuuksien välillä on tällöin funktionaalinen, ajallinen ja avaruudellinen suhde. *Kokoelmat* puolestaan muodostuvat elementeistä, jotka ovat suhteessa kokonaisuuteen samalla tavalla. Lambrix [2000] havainnollistaa laivastoesimerkillä hyvin kompleksien ja kokoelmien eroa: laivasto voidaan ajatella kokoelmana, jos sen jäsenillä ei ole mitään erikoista roolia. Se voidaan ajatella myös kompleksiksi, jos jokaisella laivalla on tietty sijainti sekä toiminta laivastossa. *Massat* ovat homogeenisiä ja ne voidaan jakaa suureisiin. Jaottelussa tunnistetaan vielä kaksi tapaa erottaa osat kokonaisuudesta. Erottelu on jaettu segmentteihin ja osuuksiin. Esimerkkinä segmentistä on artikkelin johdanto, joka on siis segmenttinä koko artikkelissa. Osuudella tarkoitetaan tässä tilanteessa puolestaan kokonaisuuden jotain osaa, joka voidaan valita ominaisuusulottuvuuden avulla. Esimerkkinä tästä on rosoiset kohdat seinästä, jolloin rosoiset kohdat ovat tietty osuus koko seinästä.

2.3. Osa-kokonaisuussuhteessa huomioitavia rajoituksia

Käsitteellisessä mallintamisessa ja tietokantojen toteutuksessa tulee ottaa huomioon osa-kokonaisuussuhteen luonteesta johtuvia rajoituksia, kuten *riippuvuus (dependency)*, *poissulkevuus (exclusiveness)* ja *monikertaisuus (multiplicity)* [Halper *et al*, 1998;Motschnig-Pitrik and Kaasboll, 1999].

2.3.1. Riippuvuus

Riippuvuus osien kesken tarkoittaa sitä, että jos jokin osa A on riippuvainen B:stä, niin A ei voi olla olemassa, jos B:tä ei ole olemassa. B voi tässä tapauksessa olla joko A:n osa tai kokonaisuus. Riippuvuuden suunta voi olla kokonaisuudesta osiin, osista kokonaisuuteen tai kaksisuuntaista. Jos esimerkiksi kirja poistetaan tietokannasta, on luonnollista, että myös sen sisältämät kappaleet ja sisällysluettelo poistetaan. Tässä tapauksessa osien olemassaolo riippuu kokonaisuuden olemassaolosta. Jos taas polkupyörän runko poistetaan, polkupyörää ei enää kokonaisuutena ole olemassa. Tässä tapauksessa kokonaisuus on riippuvainen osastaan. Suunnittelussa täytyy myös ottaa huomioon, onko tarkoituksenmukaista tuhottaessa kokonaisuutta poistaa sen kaikki osat, vai voivatko esimerkiksi kokonaisuutta edustavan polkupyörän osat jäädä olemaan, vaikka pyörä tuhottaisiinkin.

2.3.2. Poissulkeutuvuus

Poissulkeutuvuusominaisuus tarkoittaa sitä, voiko osa olla useamman kuin yhden kokonaisuuden osana vai ei. Kun on kyse konkreettisista esineistä, osaan

liittyy usein poissulkeutuvuusominaisuus. Esimerkiksi moottori voi olla osana vain yhdessä autossa kerrallaan. Kuitenkin esimerkiksi sama artikkeli voi esiintyä useammassa konferenssijulkaisussa samaan aikaan. Moottorin oleminen jonkin auton osana sulkee pois siis sen mahdollisuuden, että se on moottorina toisessa autossa samalla ajan hetkellä. Artikkelin oleminen osana jotain konferenssijulkaisua ei sen sijaan sulje pois artikkelin mahdollisuutta olla jonkin toisen konferenssijulkaisun osana. Poissulkeutuvuusominaisuuden toteutumisesta tulee pitää huolta, jotta tietokanta pysyisi johdonmukaisessa tilassa.

2.3.3. Monikertaisuus

Mallinnettaessa osa-kokonaisuussuhteita voidaan asettaa rajoituksia sille, kuinka monta osaa tietyllä oliotyypillä voi olla. Esimerkiksi tietokoneen emolevy saattaa olla sellainen, johon on mahdollista kytkeä yksi tai kaksi prosessoria kiinni, jolloin kaikki muut määrät on kielletty. Tällaisia rajoituksia kutsutaan eheysrajoituksiksi ja niitä käytetään muissakin suhdetyypeissä, kuten *omistus-* ja *liitossuhteissa.* Näitä muita suhdetyyppejä esitellään tarkemmin seuraavassa kappaleessa.

2.4. Osa-kokonaisuussuhde verrattuna muihin mallintamisrakenteisiin

Is-a –suhde [Rich and Knight, 1991; Paton *et al.,* 1996; Belford and Santone, 1989; Kim *et al.,* 1987*]* on toinen keskeinen mallintamishierarkia part-of suhteen ohella. Tällä suhteella määritellään erikoistamisia jollekin yleiselle oliotyypille/käsitteelle. Esimerkiksi käsite auto voidaan erikoistaa käsitteiksi kuormaauto, henkilöauto ja linja-auto. Erikoistamisen ideana on liittää attribuutit ja toiminnallisuus is-a -hierarkian yleisistä käsitteistä niistä erikoistettuihin käsitteisiin. Erikoistetuilla käsitteillä on lisäksi joitakin sellaisia ominaisuuksia, joita yleisimmillä käsitteillä ei ole. Is-a -suhdetta kutsutaan myös usein monissa ohjelmointi- ja mallintamisparadigmoissa periytymishierarkiaksi. Joissakin tapauksissa periytymistä voidaan rajoittaa erilaisilla suojausmäärityksillä. Toisin sanoen estetään joidenkin ominaisuuksien tai metodien periytyminen. Tämä on mahdollista esimerkiksi C++ -ohjelmointikielessä. Erikoistettavan käsitteen pitää olla semanttiselta luonteeltaan samankaltainen kuin yleisempi käsite. Valitettavasti mallintamistilanteissa tätä periaatetta saatetaan rikkoa. Esimerkiksi matemaattisia kuvioita mallinnettaessa neliöstä voidaan erikoistaa kolmio. Tämä edellyttää, että geometriset kuviot esitetään olioina siten, että niillä on attribuutteinaan tarvittavat koordinaattipisteet. Tällainen erikoistaminen voidaan tehdä, sillä neliössä tarvitaan attribuutteja kahteen, kun taas kolmiossa kolmeen

koordinaattipisteeseen. Tämä ei ole kuitenkaan käsitteellisen mallintamisen kannalta oikeaoppista, sillä neliö ei ole kolmion yleiskäsite.

Voidaan yleisesti sanoa, että is-a –suhde on tarkoitettu mallintamaan yleistys / erikoistamissuhteita, joissa attribuutit ja metodit periytyvät yleisestä erikoiseen. Erikoistetut käsitteet ovat merkitykseltään samankaltaisia kuin yleisemmät käsitteet, mutta niillä on omia uusia ominaisuuksia tai toimintoja.

Is-a ja part-of -mallinnussuhteiden lisäksi on olemassa muun muassa *attribuuttisuhde, omistussuhde* ja *liitossuhde* [Winston *et al.*, 1987]. Nämä saattavat joskus vaikuttaa part-of suhteelta, mutta eivät kuitenkaan ole sitä. *Attribuuttisuhteella* kuvataan objektin ominaisuutta, eikä attribuutilla tulisi kuvata osaa objektissa [Wand *et al.*, 1999; Artale *et al.*, 1996]. *Omistussuhde* puolestaan kuvaa objektien välillä olevaa omistussuhdetta. Toinen objekti on tällöin omistaja ja toinen omistettava. Suhde voi kuvata esimerkiksi auto-omistaja tai isä-poika suhdetta. Omistussuhteeksi voidaan virheellisesti ajatella myös polkupyörän ja renkaan välistä suhdetta. Jos kuitenkin tarkoitetaan, että polkupyörällä on rengas komponenttina, niin polkupyörä ei tällöin omista rengasta, vaan kyse on part-of –suhteesta. *Liitossuhteella* kuvataan jonkin asian liittymistä johonkin toiseen asiaan. Tällöin suhde voi olla part-of –suhde, mutta ei usein ole sitä. Esimerkiksi laukun ja ihmisen, tai korvarenkaan ja korvan suhdetta voidaan kuvata liitossuhteella, vaikka laukku ei ole ihmisen, eikä korvarengas korvan osa. Liitossuhde voi kuitenkin olla tietyissä tilanteissa samaan aikaan myös part-of –suhde. Näin on esimerkiksi polkupyörän ja renkaan part-of –suhteen kohdalla.

Is-a- ja part-of -suhteet eroavat toisistaan monella tapaa, vaikka joskus ne sekoitetaankin toisiinsa. Osa-kokonaisuussuhteessa esiintyvillä komponenteilla on toisistaan poikkeava semanttinen luonteensa, jota ei voida is-a –suhteella mallintaa. Osat tai kokonaisuudet eivät ole yleensä toistensa erikoistapauksia, joten attribuutit eivät usein periydy kokonaisuudelta osille. Tosin aikaisemmin esitetyllä osa-kokonaisuussuhdetyypillä, *lohko/kokonaisuus*, voi lohkolla olla usein samoja attribuutteja kuin kokonaisuudellansakin. Tutkielmassa tarkoitan osa-kokonaisuussuhteessa periytymisellä myös ekstensionaalisten arvojen johtamista joistakin muista part-of suhteessa olevista attribuuteista. Is-a –suhteessa periytyminen tapahtuu ainoastaan intensionaalisella tasolla. Part-of -suhteessa yleisempää sen sijaan on, että myös attribuuttien arvot periytyvät. Periytyminen tapahtuu tällöin ekstensionaalisella tasolla. Tämä ei silti välttämättä tarkoita, että perivällä objektilla olisi yhtään samaa attribuuttia. Osa-kokonaisuussuhteessa periytyminen voi lisäksi tapahtua kahteen suuntaan. Tällöin osat voivat periä joitakin ominaisuuksia kokonaisuudeltaan tai kokonaisuus puolestaan voi periä ominaisuuksia yhdeltä tai useammalta osaltaan [Halper *et al.*, 1998]. Part-of -suhteen ekstensionaalisella tasolla periytyminen

voidaan jakaa kahteen periytymisen tapaan, jos kokonaisuuden jokin arvo riippuu sen osien ominaisuuksista: perintä voi olla *suoraa perimistä* ja *emergenttistä perintää* [Goldstein *et al.*, 1999]. *Suoraa perimistä* edustaa esimerkiksi tilanne, jossa osien sijainti on sama kuin niiden kokonaisuuden sijainti. Tällöin siis jokaiselle osalle periytyy kokonaisuuden sijainti -attribuutin arvo suoraan. Emergenttinen perintä tarkoittaa sitä, että kokonaisuuden tietty ominaisuus riippuu sen osien ominaisuuksien arvoista jollakin monimutkaisella tavalla. Tästä esimerkkinä auton huippunopeus, joka riippuu sen monien osien ominaisuuksista, kuten renkaista ja moottorista. Tässä tutkielmassa emergenttisiä ominaisuuksia kutsutaan johdetuiksi attribuuteiksi.

3. Lähestymistavat osa-kokonaisuussuhteen esittämiseen ja käsittelyyn

3.1. Intensionaalisen ja ekstensionaalisen tason määrittely

Ennen kuin esittelen olemassaolevia lähestymistapoja, tarkennan edellä esitettyjä intensionaalisen- ja ekstensionaalisen tason luonnehdintoja. Tarkastelen kyseisiä tasoja relationaalisessa ja olio-orientoituneessa mallintamisessa.

Intensionaalinen eli kaaviotaso määrittää ekstensionaalisen eli ilmentymätason yhteiset piirteet. Intensionaalinen taso edustaa kaikkia tietokannassa/reaalimaailmassa olevia ilmentymiä. Ekstensionaalinen taso on puolestaan organisoitu rakenteellisesti kaavion mukaisesti. Ilmentymätaso koostuu intensionaalisella tasolla määritellyn rakenteen mukaisista esiintymistä. Ilmentymällä on tila ja mekanismi, jolla se yksilöidään, jotta se voidaan erottaa muista ilmentymistä. *Tila* tarkoittaa ilmentymään liittyvien ominaisuuksien arvojen joukkoa tiettynä ajankohtana. Päivitys on tämän tilan muuttamista.

Nämä käsitteet ovat erityisen keskeisiä käsitteellisessä mallintamisessa. Relaatiomallissa intensionaalinen taso muodostuu relaatiokaaviosta, joka määrittelee relaation rakenteen. Taso ilmaisee relaation ja attribuuttien nimet sekä attribuuttien arvoalueet. Relaation ilmentymätaso relaatiomallissa puolestaan koostuu *tupleista.* Tuplet ovat relaation attribuutteihin liittyvä arvojoukko, jossa jokaiseen relaation attribuuttiin liittyy yksi arvo. Tuplea voidaan ajatella visuaalisesti myös rivinä, jossa yksittäinen rivi ilmaisee erään ilmentymän. Rivin alkioina on siis attribuuttien arvot samassa järjestyksessä, kuin ne ovat relaatiossakin esitetty. Ilmentymien yksilöimisessä käytetään *avainta,* joka koostuu yhdestä tai useammasta tuplen attribuuttien arvoista. Avaimen tulee olla yksikäsitteinen. Millään toisella ilmentymällä ei saa olla samaa avainta samalla ajan hetkellä. Tämänkaltaista lähestymistapaa, jossa tietojen identifiointi perustuu joidenkin tietojen arvoihin, kutsutaan arvo-orientoituneeksi lähestymistavaksi. Täten relaatiomalli on arvo-orientoitunut.

Olio-orientoituneissa tietokannoissa, kuten olio-ohjelmoinnissakin, intensionaalinen taso määritellään luokkien avulla. Luokkien välillä voidaan määritellä periytymis- ja osa-kokonaisuushierarkioita. Luokissa määritellään attribuutit ja niiden arvoalueet. Lisäksi luokissa määritellään käyttäytyminen metodien avulla. Ekstensionaalinen taso koostuu olioista, jotka on perustettu luok-

kahierarkian luokkiin. Oliolla on attribuutteihin liittyvät arvot, jotka määräävät olion tilan. Oliolla on lisäksi olioidentiteetti, jonka perusteella olio yksikäsitteisesti yksilöidään. Usein olio-orientoituneet järjestelmät generoivat olio-identiteetin automaattisesti.

Kompleksisille rakenteille on ehdotettu erilaisia tietämyksen esittämistapoja. Perinteinen ensimmäiseen normaalimuotoon pohjautuva relaatiomalli (1NF) ei tue mutkikasta rakenteellista esittämistä [Liu, 1999], koska siinä relaation attribuuttien tulee olla atomisia, eli attribuuttien arvot ovat rakenteettomia. *Vierasavaimeksi* kutsutaan jonkun toisen relaation avaimen esiintymistä toisen relaation attribuuttina/attribuutteina. Kokonaisuudet ja osat joudutaan liittämään toisiinsa vierasavaimien avulla. Kokonaisuutta kuvaavassa relaatiossa varataan attribuuttikenttä jokaista välitöntä osaa kohden, jotta osan yksilöivä vierasavain voidaan sijoittaa siihen. Jos kokonaisuuden osilla on edelleen osia, niille joudutaan omissa relaatioissaan myös varaamaan attribuutit niiden osia varten. Samaa menetelmää toistetaan, kunnes koko osa-kokonaisuushierarkia on mallinnettu. Osaa kuvaava vierasavain osoittaa siis johonkin toiseen relaatioon tupleen, johon osa on mallinnettuna. Tämän vuoksi kompleksinen rakenne joudutaan hajauttamaan moniin relaatioihin, koska jokainen osa on oma relaationsa. Tämän tekee osa-kokonaisuussuhteen käsittelystä vaikeaa ja epäintuitiivista. Päivitysoperaatiot ja rakenteen käsittely hankaloituu. Relaatioalgebra on relaatiomallin yhteydessä käytetty formaali malli tiedon hakuun relaatioista. Perusrelaatioalgebrassa ei ole transitiivista sulkeumaa laskevaa primitiiviä, jolloin monimutkaisten rakenteiden läpikäynti ei yleisesti onnistu. Lisäksi johdettujen arvojen mallintaminen relaatiomallilla ei onnistu, koska käytössä ei ole mitään mekanismia tähän tarkoitukseen. Transitiivisten suhteiden käsittelyyn onkin ehdotettu relaatiomallin yhteydessä alfa-laajennettua relaatioalgebraa, jonka alfa-operaatio kykenee yleistetyn transitiivisulkeuman käsittelyyn [Agrawal, 1987]. Tämä ei kuitenkaan pysty poistamaan muita edellä esitettyjä relaatiomallin ongelmia.

3.2. Nf^2 (Non-first normal form) -esitystapa

Perinteisen relaatiomallin attribuuttien rakenteettomuuden ongelman ratkaisemiseksi on ehdotettu nf^2 –relaatiomallia [Roth *et al.*, 1988]. Tässä mallissa relaatiot sisältävät attribuutteinaan sekä atomisia että relaatioarvoisia attribuutteja. Edelleen relaatioarvoisilla attribuuteilla voi itsellään olla relaatioita attribuutteinaan ja niin edelleen. Koska malli mahdollistaa rajattoman määrän sisäkkäisiä relaatioita, sillä kyetään mallintamaan myös kompleksisia rakenteita [Niemi and Järvelin, 1996]. Tällöin ylimmän tason relaatio sulkee sisäänsä sen kaikki komponentit. Tämä tapa tiedon mallintamiseen ja käsittelyyn osa-

kokonaisuussuhteiden yhteydessä on huomattavasti intuitiivisempaa, kuin normaalin relaatiomallin yhteydessä.

Nf^2 –relaatiomallille on kehitelty sekä formaaleja että loppukäyttäjälle tarkoitettuja kyselykieliä [Korth and Roth, 1987]. Helppokäyttöiset kielet ovat SQL-kyselykielen kaltaisia. Nf^2 –relaatiomallin kyselykielillä on samat operaatiot kuin normaalilla relaatioalgebrallakin. Tämän lisäksi useissa kielissä on kaksi uudelleenstrukturointioperaatiota, nest ja unnest. Nest -operaatio lisää yhden uuden hierarkiatason nf^2 –relaatioon. Operaatiolle annetaan parametriksi joukko attribuutteja ja se muodostaa näistä attribuuteista uuden relaatioarvoisen attribuutin. Unnest -operaatio on vastakkainen operaatio nest-operaatiolle. Se poistaa yhden hierarkiatason nf^2 –muotoisesta relaatiosta. Nf^2–relaatiolla on siis paremmat mahdollisuudet tiedon strukturointiin, vaikka normaali relaatiomalli pystyykin tallettamaan saman informaation. Nf^2–relaatiolla pystytään siis mallintamaan komponentit attribuutteihin, jolloin tiedon esitys ei ole niin hajanaista. Tämä tekee kokonaisuudesta helpomman hallita, koska komposiitit ja komponentit ovat samassa rakenteessa.

Tarkastellaan seuraavaksi yksinkertaista kaaviota: *OPISKELIJA(NIMI, KURSSI)*, joka on opiskelijasta tehty nf^2–relaatiokaavio. Kaaviossa *OPISKELIJA* on relaation nimi. Relaatiossa oleva *NIMI* on atominen attribuutti, ja *KURSSI*-attribuutti on relaatioarvoinen attribuutti muotoa *KURSSI(KNIMI, ARVOSANA)*. Ekstensionaalisella tasolla tämä tarkoittaa sitä, että jokaista opiskelijan nimeä kohti voi esiintyä useita *KNIMI-ARVOSANA* -pareja, jotka yhdessä muodostavat opiskelijan nimeen liittyvän relaation ilmentymän.

Nf^2 –muotoiseen relaatiomalliin perustuvat kyselykielet ovat tavallisesti SQL-kielen laajennuksia, ja niitä on ehdotettu kompleksisten rakenteiden käsittelyyn soveltuviksi kyselykieliksi. Niitä on kuitenkin suhteellisen vaikea käyttää, koska käyttäjän täytyy ilmaista uudelleenstrukturointi -operaatiot. Tämän vuoksi monimutkaisia rakenteita sisältävien objektien käsittelyssä kyselyn muodostaminen on vaikea loppukäyttäjälle. Kyselyjen tekeminen muistuttaa usein enemmänkin proseduraalista ohjelmointikieltä, kuin deklaratiivista kyselyjen tekemistä. Tähän ongelmaan on tarjottu Timo Niemen ja Kalervo Järvelinin toimesta ratkaisua, jossa käyttäjän ei tarvitse huolehtia itse uudelleenstrukturointioperaatiosta [Niemi and Järvelin, 1996]. Heidän Prolog-toteutuksessaan käyttäjä määrittelee tuloksen rakenteen nf^2 –mallin mukaisesti ja kertoo, mistä relaatioista vastausta haetaan, sekä mitkä ehdot tulokseksi tulevan nf^2–relaation tulee täyttää. Kuitenkaan tämäkään ratkaisu ei pysty välttämään tiettyjä ongelmia, joita tämän tutkielman esittelemässä kyselykielessä pyritään välttämään. Nf^2 –relaatiomalliin pohjautuvan kyselykielen käyttäjällä täytyy olla tietoa sekä osa-kokonaisuussuhteen rakenteesta, että attribuuttien ja relaatioiden

nimistä. Lisäksi rakenteeseen eli intensionaaliseen tasoon kohdistuvia kyselyjä ei tueta riittävästi. Osa-kokonaisuussuhteelle luonteenomaista arvojen periytymistä nf^2–malli ei tue lainkaan. Nf^2-mallin heikkoutena on myös mallintamisen mahdottomuus tilanteessa, jossa komposiitissa esiintyy useampia tyypiltään samanlaisia osia. Tämä johtuu siitä, että malli ei salli kahta saman nimistä relaatioarvoista attribuuttia nf^2–relaatiossa. Tämä on heikkous osa-kokonaisuussuhteen mallintamisessa, sillä usein löytyy tarvetta tämänkaltaisten rakenteiden muodostamiseen. Esimerkiksi henkilöautossa on neljä rengasta, joiden rakenteet ovat usein samankaltaisia keskenään.

3.3. Olio-orientoituneisuus

Toinen lähestymistapa osa-kokonaisuussuhteen mallintamiseen tietokannassa on olio-orientoituneisuus. Olio-orientoituneen esitystavan periaatteena on se, että mikä tahansa todellinen tai abstrakti asia voidaan esittää oliona. Tämän vuoksi olio-orientoitunut paradigma tarjoaa luonnollisen ja ilmaisuvoimaisen tavan monimutkaisten objektien esittämiselle. [Nahouraii and Petry, 1991]. Oliomallilla on tästä samasta syystä luontevaa esittää osa-kokonaisuussuhteita sisältäviä rakenteita. Tätä tukee oliomallin ominaispiirteet: olioidentiteetti, metodit, tiedon kapselointi sekä attribuuttien mahdollisuus olla myös olioita tai oliojoukkoja [Paton *et al.*, 1999]. Olioidentiteetti huolehtii yksilöinnistä automaattisesti. Tällöin komponentti voidaan ajatella yhtä itsenäiseksi yksiköksi kuin sen kompositiokin. Metodeilla voidaan huolehtia edellä kuvailemani monimutkaiset, ekstensionaalisella tasolla tapahtuvat arvojen periytymiset. Metodeilla voidaan myös kapseloida tietoa. Tämä tarkoittaa sitä, että käyttäjältä voidaan piilottaa hänen kannaltaan epäoleellista informaatiota. Myös arvojen periytymismekanismi osa-kokonaisuussuhteiden kohdalla voidaan kapseloida käyttäjältä. Hänen ei siis välttämättä tarvitse aina tietää, kuinka ominaisuudet periytyvät osista tai kokonaisuuksista. Edellä mainituista syistä johtuen olio-orientoitunutta lähestymistapaa hyödynnetään myös tämän tutkielman kyselykielen toteutuksessa.

3.3.1. O_2-oliomalli ja OQL kyselykieli

O_2-malli [Lecluse *et al.*, 1988; Deux *et al.*, 1990] edustaa tyypillistä oliotietokantamallia. Siinä on olio-orientoituneisuuden keskeisimmät käsitteet käytettävissä, kuten objektien identiteetti, tyyppi, periytyminen, ylikuormitus ja myöhäinen sidonta [Paton *et al.*, 1996]. O_2 oliomallissa olion rakenne on lyhyesti selitettynä seuraava: Oliot rakentuvat oliotunnisteesta ja attribuuttien arvoista. Arvot voivat olla

1. atomisia (esimerkiksi merkkijonoja, luonnollisia lukuja, jne.),

2. tuplerakenteisia (tuplen alkioiden järjestyksellä on merkitys) ,
3. joukkorakenteisia (joukon alkioiden järjestyksellä ei ole merkitystä).

Olioiden esittämisessä käytetään O_2-mallissa yllä esitettyjä konstruktoreita (kuten joukko ja tuple) tietojen rakenteistamiseen. Olio on pari (i, v), missä i on olioidentiteetti, ja v on olion arvo. Olio-orientoituneisuuden yhteydessä puhutaan usein literaalista. Literaaliksi kutsutaan arvoa, jolla ei ole minkäänlaista rakennetta, eli se viittaa vain itseensä eli tietyn tyyppiseen arvoon. Literaalilla ei ole myöskään olioidentiteettiä. Esimerkiksi arvo 50 on literaali, sillä se ei koostu mistään muusta kuin arvosta 50. Literaalin arvo on siis atominen.

Tuple-rakennetta havainnollistaa seuraava esimerkki, joka kuvaa Nokia-merkkistä rengasta, jonka halkaisija on 70 ja paino 0.5. Renkaan olioidentiteettinä on oid2. Arvojen liittäminen olioidentiteetin kanssa merkitsee siis olion luomista:

(oid2, <halkaisija: 70, paino: 0.5, merkki ”Nokia”>)

Seuraava esimerkki on joukkorakenteinen, jossa olion identiteetti liitetään kahden muun olion identiteetin kanssa. Siinä olevan joukon voidaan kuvitella esittävän kahta rengasta.

(oid3, {oid1,oid2})

Yllä oleva esimerkki voisi kuvata rekka-auton rengasparia, jotka liittyvät samaan akseliin. Toisella renkaalla olisi tällöin *oid1* ja toisella *oid2* olioidentiteettinä. Joukkorakennetta kannattaa käyttää tähän tilanteeseen, jos on merkityksetöntä asettaa renkaita järjestykseen. Jos sen sijaan olisi tärkeää tietää, kumpi rengas on vasemmalla puolella ja kumpi oikealla, olisi renkaat pitänyt mallintaa tuple-rakenteen avulla.

Edellä mainituilla kahdella konstruktorilla, tuplella ja joukolla, voidaan esittää kompleksisia objekteja. Tämä johtuu siitä, että konstruktoreita voidaan käyttää sisäkkäisesti ja niissä voidaan käyttää arvoina olioidentiteettejä. Esimerkin vuoksi mallinnan seuraavaksi polkupyörän pelkistetysti O_2-mallilla. Polkupyörällä on attribuutteina hinta ja merkki. Osina ovat ohjaus, runko ja takaosa:

(oid1, <hinta: 500, merkki: ”Helkama”, ohjaus: oid3, runko: oid4, takaosa: oid5>).

Tällöin oid3 olisi sellaisen olion identiteetti, jolla olisi komponenttinaan etuhaarukka ja rengas. Oid4-olioidentiteetti puolestaan kuuluisi oliolle, joka olisi tuplerakenteinen. Tällä rakenteella kuvattaisiin rungon attribuutit. Oid5:een liittyvä rakenne olisi joukkorakenteinen. Joukon arvo sisältäisi takaosan alikomponentit. Polkupyörä itse (oid1) on tuplerakenteinen. O_2-oliomallilla voidaan luonnehditulla tavalla kuvata kompleksisia rakenteita, kuten osa-kokonaisuussuhteita. Metodien avulla voidaan lisäksi toteuttaa johdetut ominaisuudet osa-kokonaisuussuhteessa. Mallin pohjalta ei kuitenkaan kyetä toteuttamaan kyselykieltä, joka täyttää tämän tutkielman vaatimukset käyttäjäystävällisyydestä.

O_2-oliomalliin pohjautuen on tehty kyselykieli nimeltä OQL [Cluet, 1998], jolla pystytään tekemään kyselyjä mallin mukaiseen oliotietokantaan. Kyselykieli pystyy käsittelemään kompleksisia objekteja, mutta käyttäjäystävällisten kyselyjen tekeminen monimutkaisissa kyselyissä on tämän tutkimuksen vaatimuksiin usein riittämätöntä. Kyselykieli perustuu *iteraattoreiden* soveltamiseen ja osa-kokonaisuushierarkiassa navigointiin. Iteraattoreiden avulla voidaan käsitellä jonkin luokan kaikki ilmentymät käyttämällä siihen viittaavaa muuttujaa. Tällöin muuttujan voidaan katsoa edustavan mitä tahansa luokkaan kuuluvaa oliota. Iteraattorin käyttämistä esittelen seuraavalla polkupyöräesimerkillä. Kielen varatut sanat ovat lihavoituja. Jos halutaan saada kaikkien kuvitteellisessa tietokannassa olevien polkupyörien hinnat selville, muodostetaan seuraava kysely:

select p.hinta **from** p in Polkupyörä

Ilmauksessa **from** arvottaa muuttujan *p* vuoron perään kaikilla luokan *polkupyörä* ilmentymillä ja **select** valitsee jokaisen *hinta*-attribuutin arvon tulokseen. Käyttäjän täytyy ymmärtää, että *p* arvotetaan eri olioilla iteraattorissa, eli iteraattorilla käydään kaikki luokan oliot läpi. Jos edellä esitetystä pyöräesimerkin mukaisesta rakenteesta haluttaisiin tietää niiden pyörien merkit, joiden eturenkaiden halkaisija on 0.5, täytyisi muodostaa seuraavanlainen kysely:

select p.merkki **from** p in Polkupyörä **where** p.ohjaus.rengas.halkaisija = 0.5

Kyselyn perusosat ovat samankaltaisia kuin relaatiotietokantojen yhteydessä laajassa käytössä olevassa SQL kyselykielessäkin. *Select*-osassa kerrotaan, mitä halutaan, eli esimerkin tapauksessa oliota tarkoittavan *p*-muuttujan *merkki*-attribuutin arvo. *From*–osassa ilmaistaan, että *p*:n tulee olla olio polkupyöräluokassa. *Where*-osa puolestaan ilmaisee ehdon, jonka olion *p* tulee täyttää. Ilmaisussa *p*:llä viitataan mihin tahansa olioon luokassa Polkupyörä. *Where*-osassa olevassa ilmaisussa ”p.ohjaus.pyörä.halkaisija” alleviivattu osa on polkuesitys. Siinä ilmaistaan, että halkaisija-attribuuttiin päästään polkupyörän ohjauskomponentissa komponenttina olevan pyörän kautta käsiksi. Käyttäjän täytyy siis osata navigoida rakenteessa. Tämä tarkoittaa sitä, että hänen täytyy tietää tarkalleen komposiitin rakenne attribuutteineen, jotta hän voisi ilmaista tiettyyn komponenttiin johtavan polun. Jos osa-kokonaisuusrakenne on monimutkainen, navigointi tekee kyselyistä vaikeita. Pelkästään iteraattorin idean ymmärtäminen on hankalaa ja rajoittavaa. Lisäksi jos oltaisiin kiinnostuneita eri tyyppisistä polkupyöristä, kyselyä ei voitaisi ulottaa useampaan tyyppiin yhtä aikaa. Tämä johtuu siitä, että rakenne on usein erilainen eri polkupyörissä. Tällöin *halkaisija*-attribuuttiin johtava navigointipolkukin olisi erilainen eri tyypeillä. Monimutkaisissa kyselyissä voidaan lisäksi joutua soveltamaan sisäkkäisiä iteraattorirakenteita. Tällöin ilmaisu muuttuu huomattavasti vaikeammaksi. Havainnollistan sisäkkäisiä iteraattoreita sisältäviä kyselyjä esimerkein Qal-tietomallia ja -kyselykieltä käsittelevässä kappaleessa. Navigoinnin ongelma on yleinen oliomallissa. Muutkin oliomalliin pohjautuvat kyselykielet vaativat käyttäjältä tietoa kompleksisten olioiden rakenteesta ja näissä olevista attribuuteista. [Carey *et al.*, 1988].

3.3.2. Qal-tietomalli ja -kyselykieli

Joissakin oliomalleissa käytetään funktionaalisille ohjelmointikielille ominaista ohjelmointitapaa. Tällöin kieleen sisällytetään valmiita funktioita, joiden avulla kyselyjä suoritetaan. Funktiot palauttavat tuloksen, johon voidaan edelleen soveltaa funktiota. Käyttäjän täytyy tällöin tietää funktion hyväksymät parametrit ja funktion palauttama arvo. Esimerkkinä tehokkaasta funktioita hyödyntävästä kyselykielestä tarkastelen QAL-kieltä [Savnik *et al.*, 1999]. Kieltä on myös ehdotettu kompleksisten objektien käsittelyyn. QAL on ilmaisuvoimainen kyselykieli, joka pohjautuu oliomalliin aivan kuten O^2-oliomallikin. Oliot ja luokat esitetään pääpiirteittäin samaan tapaan kuin O^2:ssa. Esimerkkinä QAL-luokan määrittelystä tarkastelen *department*-luokkaa. Siinä varatut sanat ovat lihavoituja. Luokka määritellään seuraavasti:

class department

type [dept_name:string, head:employee,staff:{employee}];

Varattua sanaa **class** seuraa luokan nimi. Varatulla sanalla **type** määritellään luokan tyyppi. Tyyppi määräytyy luokan attribuuttien ja niiden tietotyyppien mukaan. Kaarisulkujen sisällä määritellään attribuutti, joka on joukkoarvoinen. Yllä olevassa *department*-luokassa *dept_name* –attribuutti on siis merkkijonomuotoinen, head on puolestaan *empolyee* –tyyppinen olio. Se on määritelty muualla luokkana, vaikka sitä ei esitellä tässä esimerkissä. *Staff*-attribuutti koostuu puolestaan joukosta *employee*-tyyppisiä olioita.

QAL-kielellä pystytään tekemään kyselyjä osa-kokonaisuussuhteita sisältäviin rakenteisiin. Tämän lisäksi kielellä pystytään muodostamaan uusia, kompleksisia rakenteita sisältäviä olioita. Seuraavaksi esittelen termistön, jota QAL-kielessä sovelletaan funktioiden käytön yhteydessä. Funktioita eli operaatioita käytetään pelkistetysti seuraavasti: s.funktio1(p), missä *s* on operaation argumentti, funktio1 on operaation nimi ja *p* on operaation parametri. Funktiolla voi olla myös useampia parametrejä tai se voi olla parametritön. Argumentti on siis olio tai luokka, johon funktiota sovelletaan. Useista muista kielistä poiketen parametri ja argumentti ei siis tarkoita samaa asiaa. Funktio voi palauttaa toimintansa tuloksena esimerkiksi numeromuotoisen arvon tai olion. QAL:ssä on kahden tyyppisiä operaatioita: malliin perustuvia ja deklaratiivisia. Kaikki malliin perustuvat operaatiot on johdettu käsitteistä, joita on käytetty tietomallissa, johon QAL-kyselykieli pohjautuu. Ne ovat siis teknisluonteisempia kuin deklaratiiviset operaatiot ja ne on tarkoitettu deklaratiivisten kyselyjen avuksi. Malliin perustuvia operaatioita ovat esimerkiksi *val* ja *ext*. *Val* käsittelee argumenttina oliota tai luokkaa ja se palauttaa argumenttinsa arvon. Jos kyseessä on olio, arvoksi palautuu olion kaikki attribuutit ja niiden arvot. Oletetaan, että s1,p1 ja p2 ovat person-luokan olioiden identiteettejä. Person luokka on määritelty seuraavalla tavalla:

class person
type [name:string, age:int, address:string, family:{person}]

Tällöin ilmaisun *s1.val* tulos voisi olla seuraava:

[name:”Jim”,age:23,address:”Ljubljana”,family:{p1,p2}]

Luokan ollessa kyseessä val-funktio palauttaa attribuuttien nimet ja niiden tyypit. *Person*-luokan yhteydessä ilmaisu *person.val* palauttaisi seuraavan rakenteen:

[name:string,age:int,address:string,family:{person}]

Ext-metodia käytetään sitomaan luokka olioihinsa. Esimerkkinä funktion käytöstä on ilmaus ”person.ext”, joka palauttaa kaikki *person*-luokan oliot. OQL-kyselykielen [Cluet, 1998] yhteydessä *ext*-metodia vastasi ilmaus *in* (esimerkiksi *p in Polkupyörä*).

Deklaratiivisia operaatioita käytetään deklaratiivisten kyselyjen muodostamisessa. Jokainen deklaratiivinen operaatio on siis funktio, joka manipuloi argumenttejaan. Seuraavaksi esittelen muutaman deklaratiivisen operaation ja lopuksi esittelen muutaman QAL-kyselykieleen pohjautuvan kyselyn.

Select–operaatiota käytetään suodattamaan argumentistaan halutut oliotyypit. Esimerkiksi ilmaisu *s.select(p)*, jossa *s* on joko olio tai tyyppi ja *p* on totuusarvotyyppinen predikaatti, palauttaa *p*-predikaatin toteuttavan *s*:ää vastaavien olioiden osajoukon. Predikaatti p voi muodostua myös sisäkkäisistä kyselyistä.

Apply–operaatio evaluoi parametrina olevan funktioilmaisun argumenttijoukossa, johon operaatiota sovelletaan. Alla esittelen esimerkin, joka havainnollistaa operaation käyttöä. Esiteltävä kysely kuvaa joukon *student*-luokan olioidentiteettejä kyseisissä olioissa esiintyviin nimiin. Identiteettifunktio *id*:tä käytetään kyselyssä tarkoittamaan yksittäistä oppilasta kaikkien oppilaiden joukosta. Kyselyssä *applyn* argumenttina on siis yksittäinen oppilas. *Apply* käy kaikki luokan ilmentymät läpi. Tämänkaltainen operaatio on iteraattori, sillä operaattori suorittaa niin monta iteraatiokierrosta, kuin joukosta löytyy käsittelemättömiä student-luokan ilmentymiä. Ilmaisussa

student.**apply**(**id**->name)

apply palauttaa kaikkien *student*-luokkaan kuuluvissa olioissa esiintyvät nimet. Parametrinä *apply*-operaatiossa voi olla mikä tahansa operaatioilmaus. Funktionaalisten kielten tapaan funktioiden palauttamaan arvoon voi edelleen soveltaa funktioita. On siis mahdollista, että apply-operaatioon sovelletaan uutta apply-operaatiota, kuten Savnik kollegoineen [1999] seuraavassa esimerkissään esittääkin. Siinä halutaan saada tulokseksi joukko joukkoja, joissa jokainen jou-

kossa oleva joukko sisältää yhden oppilaan kaikkien kurssien ohjaajat. Kysely voidaan ilmaista seuraavasti:

```
insts_sets = student.ext.apply(id->courses).apply((id.apply(id->instructor));
```

Insts_sets edustaa esimerkissä joukoista koostuvaa joukkoa, johon vastaus palautetaan. Kysely kuvaa ensin joukon *student*-luokan olioidentiteettejä joukoksi joukkoja. Nämä sisältävät olioidentiteetin, joka viittaa kursseihin, joita oppilaat ovat opiskelleet. Seuraavassa askeleessa kysely korvaa jokaisen identiteetin sisäkkäisistä joukoista olioidentiteetillä, joka viittaa tietyn kurssin ohjaajaan. Tämänkaltainen kysely on niin mutkikas, että kyselyn suorittajalla täytyy ohjelmointitaidon lisäksi olla tarkempaa ymmärrystä sisäkkäisten funktioiden soveltamisesta.

Apply_at -operaatio on *apply*:n laajennus. *Apply*-operaatiota on laajennettu uudella parametrilla, jonka tehtävänä on toimia komponentin valitsijana. Tätä operaatiota ehdotetaan ratkaisuksi sisäkkäisten rakenteiden, eli kompleksisten objektien käsittelyongelmaan kyselykielissä. Operaatio on muotoa *apply_at(p,f)*, jossa *apply*-operaatiota on laajennettu parametrillä *p*. Se osoittaa argumenttina olevan kompleksisen rakenteen komponenttipolun, josta haluttu komponentti löydetään. Komponentille suoritetaan parametrin *f* määräämä operaatio. Näin siis komposiittina olevasta objektista löydetään haluttu komponentti. Komponentti voi olla välillinen tai välitön komponentti operaation argumentille.

Näiden QAL-kyselykielen perusprimitiivien avulla voidaan tarkastella monipuolisempaa esimerkkikyselyä, jonka Savnik kollegoineen esittelevät. Kysely suodattaa sisäkkäiset komponentit department-argumentista. Komponentti identifioidaan attribuutilla *staff*. Saatu joukko, jota Savnik kollegoineen kutsuu suodatetuksi joukoksi olioidentiteettejä, sisältää viittauksen työntekijöihin, jotka ovat vanhempia kuin 45 vuotta. Kyselyn tulokselle muodostetaan ensin rakenne seuraavasti:

```
struct {[ department_name: string, head: employee, staff: {employee} ]}
depts;
```

Luotuun *depts* –nimiseen rakenteeseen sijoitetaan siis kyselyn tulos. On siis huomionarvoista, että käyttäjän tulee tietää kyselyn tuloksen rakenne QAL:ssä. Kysely itsessään ilmaistaan seuraavasti:

depts = department.**ext**. **apply**(**id.val**).**apply_at**(staff, **select**(
id->age > 45));

Kaikkia QAL:n operaatioita en esitellyt edellä. Esittelemieni operaatioiden lisäksi QAL:ssä on muun muassa joukko-opista tuttuja operaatioita ja nf^2-mallin yhteydessä aiemmin esitellyt strukturointioperaatiot nest ja unnest. Edellä olevista esimerkeistä voi kuitenkin päätellä, että ohjelmointitaidoton loppukäyttäjä ei pysty QAL-kyselykieltä käyttämään. Kielen primitiivit eivät ole helposti omaksuttavissa, koska suuri osa niistä on iteraattoreita, jolloin iteraatiokäsitteen tulee olla käyttäjälle tuttu. Lisäksi käyttäjän tulee omaksua olioajattelussa käytettyjä käsitteitä, kuten luokan ja olion arvo, sekä olion identiteetti. Edelleen käyttäjän täytyy osata käyttää metodeja, tietää niille annettavat argumentit ja parametrit sekä huolehtia siitä, että tyypit ovat oikeita. Monet metodit saattavat toimia eri tavalla kuin käyttäjä on olettanut, jos hän antaa argumentiksi tai parametreiksi vääriä tyyppejä. Operaatioiden palauttamien arvojen kanssa tilanne on sama. Käyttäjän tulee tietää, mitä operaatio palauttaa eri tilanteissa. Palautusarvojen tyypithän voivat vaihdella riippuen operaation argumentista ja parametreista. Tietojenkäsittelytaitoja huonosti hallitsevan käyttäjän tilanne on toivoton viimeistään kyselyissä, joissa täytyy metodin palauttamalle arvolle soveltaa jotain toista metodia. Tällaiset kyselyt eivät välttämättä luonnistu virheettömästi edes kokeneemmalta kyselykielen käyttäjältä. Tässäkään kyselykielessä ei päästä eroon rakenteessa navigoinnista tai strutkturointioperaatioiden käytöstä. Toisin sanoen käyttäjän tulee tuntea tietokannassa olevien objektien rakenne. Yhteenvetona QAL-kyselykielestä voi sanoa, että se on ilmaisuvoimainen, mutta vaikea käyttää. Loppukäyttäjällä tulee olla kokemusta ohjelmoinnista.

3.3.3. ODMG 3.0 -standardi

Edellä esittelemissäni kahdessa olio-orientoituneessa lähestymistavassa osakokonaisuussuhteet on mallinnettu siten, että luokan komponentit ovat komposiitin attribuutteina. Tällöin komposiitista päästään käsiksi komponenttiin, mutta komponentista ei päästä käsiksi sen komposiittiin. Olio-orientoituneelle lähestymistavalle on yleistä tällainen mallintaminen [Junkkari, 2001]. Tästä seuraa, että komponentti ei sisällä informaatiota siitä, minkä kokonaisuuden osana se on. Moniin olio-orientoituneisiin tiedon esitystapoihin perustuen osan komposiitti voidaan löytää vain käymällä kaikki kompleksisia objekteja sisältävät rakenteet läpi komposiitista komponenttiin. Etsimistä jatketaan niin kauan kunnes löydetään komposiitti, jolla on haluttu komponentti. Jos komponentista halutaan suoraan kompositioon, tulee suhteet mallintaa jokaisessa luokassa

kaksisuuntaisesti. Seuraavaksi tarkastelen standardia, jossa suhteet mallinnetaan tällä tavalla.

ODMG:n (Object Data Management Group) tekemässä standardissa [Cattell *et al.*, 2000] tyyppi määrittelee olion rakenteen ja käyttäytymisen. Käyttäytyminen määritellään joukkona metodeita. ODMG:ssä oliotyypillä voi olla useampia välittömiä yliluokkia. Tätä kutsutaan *moniperiytymiseksi*. Luokka voi siis periä ominaisuuksia monilta luokilta. Oliot ja literaalit on eroteltu toisistaan Oliolla voi olla attribuuttien arvoja, suhteita ja metodeja. Standardissa voidaan käyttää myös *rajapintoja*. Rajapintojen käytön mahdollisuus tukee sovelluksen rakentamista ja suunnittelemista toteutuksen yksityiskohdista välittämättä. Rajapintamekanismia voi pitkälle vietynä kuitenkin kritisoida luokkien lukumäärien kasvulla [Koskimies, 2000]. Suhteiden mallintaminen tapahtuu määrittelemällä eksplisiittisesti suhteelle rajoite: suhde voidaan rajoittaa yksittäiseksi, joukkotyyppiseksi tai listaksi. Suhteen rajoitteen lisäksi määritellään suhteen kohde. Tätä seuraa jokin kuvaileva suhteen nimi. Tämän lisäksi suhteelle määritellään käänteissuhde.

Seuraavaksi esittelen esimerkkiä polkupyörän ohjaussysteemistä, jossa etupyörän ja ohjauksen välitön komponenttisuhde mallinnetaan ODMG-standardilla seuraavasti:

class steering

{

relationship wheel has_component

inverse wheel is_component

}

Lihavoidulla kirjoitetut sanat ovat varattuja. Luokan sisällä oleva ensimmäinen lause tarkoittaa, että *steering*-luokalla on *wheel*-luokka välittömänä komponenttinaan. Toisena lauseena on ensimmäisen käänteissuhde, eli *wheel*-luokka on komponenttina *steering:*ssä. Suhteen mallinnus pitää tehdä myös *wheel*-luokkaan. *Wheel*-luokassa siis määritellään suhde ilmaisulla ”**relationship** steering has_composite” sekä käänteinen suhde: ”**inverse** steering is_composite”. Jos ohjauksella voisi olla useampia renkaita, joiden keskinäisellä järjestyksellä ei ole merkitystä, **relationship** –termin sijasta käytettäisiin **relationship set** –sanaa. Jos puolestaan ohjauksella olisi useampi rengas ja niiden keskinäisellä järjestyksellä olisi väliä, käytettäisiin varattua sanaa **relationship list.**

Tämänkaltainen suhteiden mallintaminen on työlästä, koska suhteet joudutaan mallintamaan moneen kertaan. Se ei silti poista navigoinnin tarvetta,

koska ODMG:ssä käytetään samaa OQL-kyselykieltä [Cluet, 1998], kuin edellä esitetyssä O_2-oliomallissakin. Koska suhteet on mallinnettu molempiin suuntiin, navigointia voidaan harjoittaa kompositiosta komponentteihin, mutta myös päinvastoin. Luokan mallintaja voi käyttää helposti tulkittavia suhteiden nimiä, jolloin navigointi tulee käyttäjälle intuitiiviseksi. Kyselykielen käyttäjän tulee kuitenkin tietää suhteiden nimet. Tällöin ODMG:n suhteiden esitystapa ei lopulta eroa käytettävyydeltään normaalista navigoinnista, jossa tämänkaltaista esitystapaa ei käytetä. Ainoa etu suhteiden mallintamisessa ODMG:n tapaan onkin se, että kumpikin suhteen osapuoli tietää toisensa. Navigoinnin suunta on siis kyselijän päätettävissä. Tästä ominaisuudesta on käyttäjäystävällisyyden kannalta hyötyä.

Yhteenvetona olioperustaisista tietokannoista voidaan todeta, että ne tarjoavat hyvät välineet osa-kokonaisuussuhteen mallintamiseen. Olioidentiteettien avulla voidaan osat ja kokonaisuudet yksilöidä helposti. Olioidentiteettien avulla voidaan luopua siis avaimien käytöstä, joka on arvo-orientoituneessa mallintamisessa yksilöinnin mekanismina. Avaimet ovatkin turhia käsitteellisessä mallintamisessa semanttisessa mielessä ja niitä käytetään vain toteutusmekanismina [Wand, 1999]. Metodien avulla voidaan hoitaa johdettujen attribuuttien toteutus ja muita hyödyllisiä toimintoja. Oliotietokantojen heikkoudeksi voi kuitenkin laskea sen, että olioiden välisistä suhteista täytyy huolehtia kyselykielissä navigoinnin tai uudelleenstruktu-rointioperaatioiden avulla. Kokonaisuuden sisäisiä suhteita ei siis tueta oliotietokannoissa riittävästi [Beraha and Su, 1999].

3.4. Deduktiiviset tietokannat

Deduktiivisissa tietokannoissa yhdistyy relaatiotietokantaan ja logiikkaohjelmointiin liittyviä tekniikoita [Liu, 1999]. Deduktiivisilla tietokannoilla on suuri ilmaisuvoima. Säännöt määritellään logiikan sääntöjen avulla ja päättelykykynsä ansiosta deduktiivisia tietokantoja käytetään muun muassa tekoälysovelluksissa ja asiantuntijajärjestelmissä. Niillä voidaan määritellä myös transiitiivisen sulkeuma, johon normaali relaatiomalli ei kykene. Deduktiivisuus on arvo-orientoitunut lähestymistapa tiedon esittämiseen. Rakenteellisen tiedon esittäminen on helpompaa kuin relaatiomallilla, sillä osat voidaan mallintaa monimutkaisilla loogisilla termeillä. Logiikkaohjelmoinnin yhteyteen on kehitelty konstruktoreita erityyppisiin tiedon esittämisen tarkoituksiin [Niemi and Järvelin, 1991]. Niihin kuuluvat muun muuassa oliotietokantojen yhteydessä esitellyt tuple- ja joukkokonstruktorit.

Rakenteellisen tiedon esittäminen ei ole kuitenkaan yhtä helppoa kuin olio-orientoituneisuuden yhteydessä. Monimutkaisten termien käsittely ei ole help-

poa ja etenkin päivitykset ovat hankalia [Liu, 1999]. Logiikkaohjelmoinnissa ei ole myöskään selkeää tapaa intensionaalisen ja ekstensionaalisen tiedon erottelulle samassa merkityksessä kuin oliotietokantojen yhteydessä. Deduktiivisten tietokantojen yhteydessä intensionaalisella osalla tarkoitetaan tietokannan johdettuja tietoja. Johdetut tiedot määritellään logiikan sääntöjen avulla. Ekstensionaalinen taso tietokannasta koostuu tietokannassa olemassaolevasta tiedosta, johon sääntöjä sovelletaan [Niemi *et al.*, 1998; Niemi *et al.*, 2000]. Esittelen logiikkaohjelmoinnin peruskäsitteitä tutkielman myöhemmässä vaiheessa.

Deduktiivinen oliotietokanta on lupaava lähestymistapa. Siinä yhdistyy olio-orientoituneisuuden tiedon esitystapa ja logiikkaohjelmoinnin päättelykyky. Yksimielisyyttä siitä, mitä deduktiivinen olio-orientoituneisuus tarkoittaa, ei vielä ole muodostunut tiedeyhteisössä [Liu, 1999]. Näiden kahden paradigman yhdistäminen ei ole yksinkertaista, koska lähestymistavat ovat hyvin erilaiset. Deduktiivinen olio-orientoituneisuus on valittu myös tutkielmassa esiteltävän kyselykielen toteutuksen pohjalle.

4. Helppokäyttöisen ja ilmaisuvoimaisen kyselykielen piirteet

4.1. Helppokäyttöisen kielen vaatimuksia

Seuraavaksi tarkastelen helppokäyttöisen osa-kokonaisuussuhteiden käsittelyyn kykenevän kyselykielen ominaisuuksia. Ensiksikin kielen tulee pystyä käsittelemään sekä intensionaalisella, että ekstensionaalisella tasolla olevia tietoja. Intensionaalisen tiedon käsittely on välttämätöntä, jotta rakennetta voidaan analysoida tehokkaasti. Lisäksi kielen tulee pystyä yhdistämään intensionaalinen ja ekstensionaalinen kyselytyyppi yhteen kyselyyn. Intensionaalisen ja ekstensionaalisen tietojen yhdistäminen on normaali toimenpide kaikissa kyselykielissä, mutta tässä yhteydessä kysely tarkoittaa eri asiaa. Samassa kyselyssä pitää nimittäin pystyä kysymään sekä intensionaalisen, että ekstensionaalisen tason tietoja ja näillä välituloksilla yhdistämään kysely tasolta toiselle. Esimerkkinä tämänkaltaisesta kyselytyypistä on tilanne, jossa käyttäjä haluaa tietää jonkin kokonaisuuden kaikkien alikomponenttien attribuuttien arvot. Jos hän ei kuitenkaan tiedä komponenttien nimiä ja näiden attribuutteja hänen täytyy suorittaa ensin intensionaaliseen tasoon kohdistuva kysely ja tämän jälkeen vielä erikseen ekstensionaaliseen tasoon kohdistuva kysely. Jos intensionaalisten ja ekstensionaalisten kyselyjen yhdistäminen samaan kyselyyn onnistuu, hän voi suorittaa kyselyn yhdellä kyselyllä. Käyttäjäystävällisyys paranee kyselyissä, jos niissä pystytään yhdistämään molempien tasojen käsittelyt samaan kyselyyn. Tällaisia kyselyjä nimitän *yhdistetyiksi kyselyiksi*. Eri tasoja käsitteleviä kyselyitä tarkastellaan seuraavissa aliluvuissa.

Kielen käyttäjältä ei voida olettaa ohjelmointitaitoja. Hänen tulee kyetä tekemään kyselynsä määrittelemällä kyselyn lopputulos mahdollisimman korkealla abstraktiotasolla. Hänen ei siis tarvitse määritellä kyselyn lopputulosta proseduraalisesti. Tutkielmassa toteutettavassa kyselykielessä on periaatteena, että kyselyjen tekemisessä käyttäjän täytyy osata soveltaa vain jaetun muuttujan ideaa. Lisäksi hänen täytyy ymmärtää konjunktion, disjunktion ja sulkujen merkitys sekä opetella kielen primitiivien käyttäminen. Primitiivien nimien tulee olla kuvaavia, jolloin niiden käyttö on mahdollisimman intuitiivista. Primitiivien tarkoituksena on sallia kyselyissä luonnollista kieltä muistuttavat ilmaukset. Primitiivejä ei kuitenkaan saa olla liikaa, jotta niiden muistaminen ei koituisi työlääksi.

Käyttäjältä ei voida edellyttää rekursion eikä iteraation käsitteellistä sisäistämistä. Tällöin kielen primitiivien tehtävänä on piilottaa loppukäyttäjältä rekursiivisten ja iteratiivisten tilanteiden, kuten välillisten komponenttien ja komposiittien käsittelyt. Tämä ei ole itsestäänselvyys olemassaolevissa kielissä. Esimerkiksi eräät käyttäjäystävällisiksi tarkoitetut SQL-kielen laajennokset nf^2 – relaatiotietokantojen yhteydessä eivät pysty käsittelemään kuin ylintä komponenttihierarkiatasoa [Niemi and Järvelin, 1996].

Kielen tulee tarjota kaksisuuntainen käsittely komponenttien ja komposiittien välillä, kuten ODMG-standardissakin. Käsittelyssä ei kuitenkaan tule käyttää navigointia. Käyttäjän pitää saada välilliset ja välittömät komponentti ja komposiitit sekä intensionaalisella-, että ekstensionaalisella tasolla ilman navigointia. Tämä ominaisuus lisää huomattavasti kielen käyttäjäystävällisyyttä. Tämänkaltainen navigointi mahdollistaa myös rakenteellisesti toisistaan poikkeavien osa-kokonaisuussuhteiden käsittelyn samassa kyselyssä.

Kielen täytyy lisäksi tukea osa-kokonaisuussuhteelle luonteenomaista ominaisuuksien arvojen periytymistä tavalla, joka on käyttäjälle näkymätöntä. Komposiittien kohdalla kapseloinnin onkin katsottu olevan tärkeässä asemassa käyttäjäystävällisyyden kohottamisen kannalta [Civello, 1993]. Rakenteen analyysiä helpottaa myös, jos kieli tarjoaa primitiivejä osa-kokonaisuussuhteen muiden erikoispiirteiden tarkasteluun. Tällaisia helpotuksia on muun muassa ylimmän tason oliotyypin tai olioiden etsiminen. Samoin tulee pystyä hakemaan myös perustyyppi ja oliot. Polkuesityksen saaminen tietyltä oliotyypiltä antaa puolestaan hyvän kuvan siitä, missä kohtaa osa sijaitsee rakenteessa. Kyselykielen mallintamisratkaisun pitää mahdollistaa myös samantyyppisten komponenttien mallintamisen samassa rakenteessa, nf^2-mallista poiketen.

4.2. Eri tasojen tarkastelu kyselykielessä

Puhtaasti intensionaalisilla kyselyillä tarkoitetaan tässä tutkielmassa kyselyitä, joilla haetaan informaatiota ainoastaan intensionaalisesta tasosta. Tällöin siis kyselyn vastaukset kuuluvat intensionaaliseen tasoon. Intensionaaliset kyselyt ovat tärkeitä esimerkiksi tilanteissa, joissa kyseltävien kohteiden kaaviotaso on käyttäjälle entuudestaan tuntematon. Kaikki rakenteen analysointiin liittyvät kyselyt ovat intensionaalisia. Käyttäjä saattaa olla kiinnostunut esimerkiksi kyselyn kohteen välillisistä tai välittömistä komponenteista tai komposiiteista. Myös kohteen ominaisuuksien selvittäminen kuuluu intensionaalisiin kyselyihin. Käyttäjä voi intensionaalisen kyselyn avulla saada selville, mitkä ovat kyseltävän kohteen komponentit ja komponenttien attribuutit. Myös tietyntyyppiseen komponenttiin johtavien polkujen kysely kuuluu intensionaalisiin kyselyihin. Lisäksi peruskomponenttien ja ylimmän tason komponenttien selvittä-

minen tapahtuu intensionaalisella tasolla kaavion perusteella. Intensionaalisten kyselyiden mahdollistaminen helpottaa käyttäjää, kun kyselykielen avulla on mahdollista lievittää kaaviotason tietojen muistamistaakkaa. Lisäksi intensionaaliseen tasoon kuuluvat kyselyt, jossa intensionaalisen tason tietoja kysellään joidenkin ekstensionaalisten kriteerien perusteella. Jos esimerkiksi halutaan saada selville, missä oliotyypin ominaisuudessa esiintyy arvo *red*, saamme vastaukseksi kriteerin täyttäviä oliotyyppejä. Tämänkaltainen intensionaalisen tason analysointi on hyödyllinen myös käyttäjälle, jolle kaaviotaso on jo entuudestaan tuttu. Ekstensionaaliseen tasoon liittyvästä tiedosta voidaan siis jossain tilanteissa saada parempi kuva intensionaalisen vastauksen avulla [Motro, 1994].

Vaikka kyselyn vastauksena saataisiin numeroarvoja, voi kysely olla silti intensionaalinen kysely. Tällaisesta tilanteesta on esimerkkinä kysely, joissa vastaukseksi halutaan saada tietyn tyypin peruskomponenttien määrä. Tällöin arvo tarkoittaa rakenteeseen liittyvää lukumäärää.

Kyselykielen tulee tukea myös ekstensionaalisia kyselyjä. Puhtaasti ekstensionaalisilla kyselyillä tarkoitetaan tutkielmassa kyselyjä, joiden vastaukset sisältävät pelkästään ekstensionaalisen tason tietoja. Intensionaalinen taso täytyy tällöin olla käyttäjän tiedossa, jotta kyselyn tekeminen olisi mahdollista. Käyttäjän täytyy ilmaista kyselyssään attribuuttien tai luokkien nimet, joiden ekstensionaalisen tason arvoista hän on kiinnostunut. Olennaista tässä kyselytyypissä on se, että vastaukseksi tulee ainoastaan ekstensionaalisen tason tietoa, kuten attribuuttien arvoja. Tämän tapainen kyselytyyppi on tärkeä, sillä usein ollaan kiinnostuneita tietokannan kuvaamien reaalimaailmassa olevien abstraktien tai konkreettisten asioiden tiloista. Ekstensionaaliseksi kyselyksi luokitellaan myös kysely, joka antaa tiettyjen intensionaalisten kriteerien perusteella pelkästään ekstensionaalista tietoa.

Intensionaalis-ekstensionaalisilla kyselyillä tarkoitan tutkielman kielessä kyselyjä, joissa kyselyn tuloksena saadaan kumpaankin tasoon liittyviä tietoja. Tällaista kyselyä edustaa seuraava esimerkki: Kyselyssä halutaan vastaukseksi kokonaisuuden osat, osiin liittyvien ominaisuuksien nimet ja edelleen ominaisuuksiin liittyvät arvot. Tällöin vastaukseksi saadaan sekä intensionaalista että ekstensionaalista tietoa. Vastaus muodostuu tällöin osan nimestä, siihen liittyvästä ominaisuudesta ja ominaisuuden arvosta. Näistä kaksi ensimmäistä ovat intensionaalista tietoa ja viimeinen intensionaalista tasoa vastaavaa ekstensionaalista tietoa. Kysely havainnollistaa selkeästi, mihin käsitteeseen kukin arvo liittyy tietokannassa tietyllä ajan hetkellä. Intensionaalis-ekstensionaaliset kyselyt ovat tavallisia esimerkiksi SQL:ssä. Siinä saadaan suurimmassa osassa kyselyjä vastaukseksi intensionaalinen taso (relaation kaavio) ja vastaava ekstensio-

naalinen taso (relaatio). Seuraavaksi esittelen tällaisen SQL-kyselyn. SQL-kyselyssä haetaan tietoa kuvitteellisesta relaatiotietokannasta, jossa on *bicycle*-relaatio: **select** *color, price from bicycle* **where** *price > 3000*. Tämän kyselyn tuloksena saadaan tulosrelaatio, jossa attribuuttien *color* ja *price* arvot tulostuvat attribuuttien nimien alle. Attribuutit haetaan *bicycle*-nimisestä relaatiosta ja ehtona on, että vastaukseksi saatavien *price*-attribuuttien arvojen tulee olla suurempia kuin 3000. Vastaus voisi olla esimerkiksi seuraava:

color	price
brown	4500
white	3200
red	7400

Sarakkeiden niminä esitetään siis intensionaalinen taso ja rivit esittävät ekstensionaalisen tason.

Jokaisen kyselytyypin kohdalla on tässä tutkielmassa tarkasteltavassa kielessä mahdollista tehdä edellä luonnehdittuja yhdistettyjä kyselyjä. Tällöin vastaus on jokin yllämainitun kolmen kyselytyypin mukainen. Jos halutaan esimerkiksi suorittaa puhtaasti ekstensionaalinen kysely, mutta intensionaalisesta tasosta ei ole tarpeeksi tietoa, voidaan kysely suorittaa silti yhden kyselyn avulla. Kysely olisi voitu jakaa myös kahteen erilliseen kyselyyn. Tällöin tehtäisiin ensin kysely, josta kyselyyn tarvittava intensionaalinen tieto selviää. Tämän jälkeen suoritettaisiin erillinen ekstensionaalinen kysely edellisen kyselyn tietojen perusteella. Yhdistetyn kyselyn tekeminen on kuitenkin vaivattomampaa. Kahteen erilliseen kyselyyn jakaminen on työlästä erityisesti tilanteissa, joissa ensimmäinen kysely tuottaa suuren määrän vastauksia. Tällöin jälkimmäinen kysely joudutaan suorittamaan jokaiselle ensimmäisessä kyselyssä saadulle vastaukselle.

Seuraavaksi esittelen erityyppisiä kyselyjä lisää esimerkkien valossa, jotta niiden luonne ja käytännön merkitys tulisi selvemmin esille.

4.3. Esimerkkejä intensionaalisista kyselyistä

Jos esimerkiksi venekorjaamon työntekijä haluaa saada selville, missä osassa tietyssä venemallissa on pilssipumppu, hän on tällöin kiinnostunut venemalliin liittyvästä intensionaalisesta tiedosta. Pyörätehtaalla puolestaan saatetaan tarvita tietoa, kuinka monta mutteria tietyn pyörämallin komponenttiin tarvitaan, jotta niitä voitaisiin tilata sopiva määrä. Laivayhtiö voi tarvita tiedon, minkä tyyppisiä hyttejä on laivan viidennellä kannella. Vastaus voidaan haluta vielä

siten, että eri hyttityypit on jaoteltu erikseen ja tyyppejä vastaavat määrät ovat tyyppien vieressä. Kahdessa viimeisessä kyselyssä saadaan arvoja vastaukseksi. Kummassakin tapauksessa arvot on kuitenkin tuotettu intensionaaliseen tietoon perustuen.

4.4. Esimerkkejä ekstensionaalisista kyselyistä

Monesti riittää tieto pelkästään kohdealueen ekstensionaaliseen tasoon liittyvistä seikoista. Esimerkiksi venekauppias saattaa tarvita tiedot kaikista veneen sohvaverhoilukankaan mahdollisista väreistä, joita kyselyn ajankohtana on saatavilla. Pyöräkauppias voi puolestaan haluta tietokannastaan tulostettavaksi kaikkien saatavilla olevien renkaiden koot. On hyvä huomata, että vaikka kauppias on edellä kiinnostunut pelkästään ekstensionaalisesta tasosta, hänen täytyy määritellä kyselytyyppi intensionaalisten käsitteiden avulla. Käsitteet ovat joko tyyppejä tai tyyppien ominaisuuksia. Jotta voisimme esimerkiksi saada kyselykielen avulla tulostetuksi kaikkien kannassa olevien autojen hinnat, meidän täytyy luonnollisella kielellä ilmaistuna esittää asia seuraavasti: ”Anna kaikki auto-tyyppisten olioiden hinta-attribuuttien arvot”. Tällöin tyyppi *auto* ja siihen liittyvä attribuutti *hinta* täytyy olla kyselyn suorittajan tiedossa jo ennen kyselyn suorittamista.

4.5. Esimerkkejä intensionaalis-ekstensionaalisista kyselyistä

Olemassaolevat kyselykielet antavat useimmiten intensionaalis-ekstensionaalisia vastauksia. Ensimmäinen esimerkki tämäntyyppisestä kyselystä oli aikaisemmin esitelty SQL-kysely. Toisena esimerkkinä tarkastelen kyselyä, jossa samalla kertaa saadaan kattava kuva osa-kokonaisuusrakenteesta ja yleiskuva ekstensionaalisesta tasosta: Tietokonekauppias haluaa saada selville kaikkien emolevyjen alikomponentit, niiden attribuutit ja attribuuttien arvot. Tällöin siis vastauksessa on sekä intensionaalista että ekstensionaalista tietoa. Kolmanneksi esimerkiksi otan analysoivan kyselyn, jossa haetaan kaikkien osa-kokonaisuussuhteesta löytyvien ylimpien komposiittityyppien mukaisten ilmentymien muoviosien yhteenlaskettu paino. Tällöin vastaukseksi tulee kaikkien kokonaisuuksien nimet, jotka ovat intensionaalista tietoa ja näitä vastaavat ekstensionaaliset arvot, jotka kuvaavat yhteenlasketun painon summan.

4.6. Esimerkkejä yhdistetyistä kyselyistä

Edellisessä kappaleessa kerrottiin intensionaalis-ekstensionaalisista kyselyistä, jotka tuottivat kummankin tason tietoa vastaukseksi. Tämän lisäksi kappaleessa toisena esimerkkinä ollut kysely oli samalla myös yhdistetty kysely. Siinä kyseltiin emolevyjen alikomponentit ja niiden attribuutit, jotka olivat siis intensionaalista tietoa. Perinteisellä SQL-kielellä toteutettuna käyttäjällä olisi kuiten-

kin täytynyt olla tieto alikomponenttien nimistä ja attribuuttien arvoista, jotta hän olisi voinut kysyä niiden ekstensionaalisen tason arvoja. Edellä olleessa esimerkissä käyttäjä kykeni suorittamaan kyselyn, vaikka rakenteelliset tiedot eivät olleetkaan selvillä.

Toisena esimerkkinä otan venekauppiaan, joka haluaa saada selville, millä veneen pilssijärjestelmän alikomponenteilla on komponentteina muoviosia. Tässäkin tapauksessa tarvitaan kummankin tason tietojen yhdistämistä. Pilssijärjestelmän komponentti on käsitteenä intensionaalinen, koska se on rakenteellista tietoa. Tieto alikomponenttien materiaalista on puolestaan ekstensionaalisella tasolla.

Seuraavassa esimerkissä havainnollistan yhdistettyjä kyselyitä mahdollistavan kielen etua erityisesti iteraatioon perustuviin kieliin verrattuna. Esimerkiksi otan lentokonevarikon, jolla on 100 lentokonetta. Suurimmalla osalla lentokoneista on kolme moottoria. Tällöin moottorit ovat tyypiltään samanlaisia, mutta kaksi niistä on kiinni siipien alla ja kolmas peräosassa. Kyseisillä kolmen moottorin lentokoneilla ei ole kuitenkaan kaikilla samanlaista rakennetta. Joissain lentokoneissa moottori on voitu mallintaa johonkin lentokoneeseen suoraan pyrstön komponentiksi. Toisessa lentokonemallissa pyrstössä on kiinni moottoriteline, johon moottori on kiinnitettynä. Kolmannessa moottori on rungon peräosassa kiinni. Varikon mekaanikko haluaa saada tietokannasta tietoa kaikkien niiden lentokoneiden takamoottoreista, jotka ovat vuosimallia 1981. Iteraatiopohjaisella kielellä, asian selvittäminen ei onnistuisi yhdellä kyselyllä, jos kyselyssä käytetään navigointia. Tämä johtuu siitä, että iteraatio käy kaikki lentokoneet läpi, perustuen samaan navigointipolkuun. Mekaanikon täytyisi tällöin selvittää ensin joka lentokoneen kohdalla takamoottoriin johtava navigointipolku.

Jos sen sijaan jokainen lentokone on mallinnettu siten, että niistä löytyy takaosa, kysely voidaan tehdä yhdellä yhdistetyllä kyselyllä. Mekaanikko voi ilmaista tällöin kielellä, että hän haluaa saada tiedot sellaisten moottorien ilmentymistä, joilla on komposiittina takaosa. Lisäksi samassa kyselyssä ilmaistaan vielä, että tällaisten moottorien vuosimallin tulee olla 1981. Tällöin selvitetään ensin intesionaalisella tasolla moottorit, vaikka niiden tarkkaa sijaintia ei tiedetä. Tämä vastaa navigointipolun selvittämistä ja on intensionaalista tietoa. Sitten rajoitetaan moottoreita siten, että ainoastaan vuosimallin 1981 moottorit saadaan vastaukseksi. Tämä on ekstensionaalista tietoa. Samassa kyselyssä saatiin siis selville ensin intensionaalinen tieto ja tämän jälkeen tätä tietoa hyväksikäyttäen kyselyssä voitiin määrittää ekstensionaaliset kriteerit ja saada ekstensionaalista tietoa.

4.7. Muita kielen vaatimuksia osa-kokonaisuussuhteiden käsittelyyn

Kielen käyttäjän tulee olla aina selvillä, mitä käsitteitä ja käsitetasoja hän kyselyissään kulloinkin käsittelee. Primitiivin nimen tulee selkeästi kuvata käyttötarkoitus ja se, mitä argumentteja primitiivi käsittelee. Monikäyttöisiä primitiivejä ei sallita. Tällä tarkoitetaan sitä, että jollekin primitiiville käyvät vain tietyn tyyppiset argumentit. Primitiivejä, joiden käyttäytyminen vaihtelee argumentin sisällöstä riippuen ei kielessä sallita. Tällaiset primitiivit voivat olla tehokkaita, mutta ne voivat myös olla harhaanjohtavia, jos niitä ei hallitse kunnolla; käyttäjä voi saada tietämättään vääriä vastauksia.

Kyselykielen ei tule myöskään sisältää QAL-kielen luonteisia-, malliin perustuvia primitiivejä. Kaikki ohjelmointiin liittyvä käsitteistö, joka ei ole olennaista kyselyjen suorittamisen onnistumisen kannalta, on piilotettava käyttäjältä näkymättömiin. Täten käyttäjän ei tarvitse huolehtia olioidentiteettien käsittelystä eikä muista toteutukseen liittyvistä piirteistä.

5. Kyselykielen edellyttämä esitystapa

5.1. Intensionaalisen ja ekstensionaalisen tason sitominen toisiinsa

Jotta intensionaalisen ja ekstensionaalisen tason sitominen toisiinsa mahdollistuisi, se edellyttää kummankin käsitetason tietojen eksplisiittistä esittämistä. Esitys tulee tehdä siten, että käsitetasolta on mahdollisuus siirtyä toiselle käsitetasolle saumattomasti. Tämän vuoksi tasot tulee pystyä integroimaan keskenään. Esitystavan tulee mahdollistaa intensionaalisen tason rakenteen kuvautuminen sitä vastaavalle ekstensionaalisella tasolla oleville ilmentymille yksikäsitteisesti ja päinvastoin. Tällöin osa-kokonaisuushierarkioissa intensionaalisella tasolla olevista luokkien ja attribuuttien nimistä voidaan siirtyä käsittelemään niitä vastaavia ekstensionaalisen tason olioita tai arvoja.

Kahden tason toisiinsa integrointiin käytetään Timo Niemen [1983] kehittämää indeksointimekanismia, jota on edelleen kehittäneet Niemi, Kalervo Järvelin ja Marko Junkkari erilaisiin tarkoituksiin kompleksisten rakenteiden mallintamisessa. Myös tämä tutkielma pohjautuu keskeisesti tähän indeksointimekanismiin. Tutkielmassa esittelemäni kieli pohjautuu erityisesti Marko Junkkarin formalismiin [2001], jossa indeksointimekanismia on sovellettu olio-orientoituneeseen esitystapaan. Formalismin yhtenä keskeisenä motivaationa on mahdollistaa kyselykieli, jonka suunnittelua ja prototyyppitoteutusta tässä tutkielmassa tarkastellaan. Tarkastelen seuraavaksi tätä formalismia.

5.2. Olio-orientoitunut esitystapa osa-kokonaisuussuhteille

Marko Junkkarin [2001] esitystapa mahdollistaa intensionaalisen ja ekstensionaalisen tason kiinteän sitomisen toisiinsa indeksointimekanismin ja joukko-opin avulla. Esitystavan tarkoituksena on myös osa-kokonaisuussuhteita sisältävien rakenteiden tehokkaan analysoinnin mahdollistaminen. Esitystapa perustuu rakenteelliseen olio-orientoituneeseen mallintamiseen. Tällöin käyttäytymisnäkökulmaan ei formalismissa oteta kantaa, joten esitystavassa olioilla ei voi olla metodeja. Teoria on toteutuskielestä ja ympäristöstä riippumatonta, koska se esitetään joukko-opin avulla.

Osa-kokonaisuussuhteita sisältävä rakenne esitetään part-of structure element (lyhyesti PSE) –rakenteena, joka sisältää kompleksisen rakenteen sekä ekstensionaalisen että intensionaalisen tason erikseen. Tiedot esitetään tavalla, joka mahdollistaa indeksien avulla tapahtuvan tasojen integroinnin. Yksi PSE-rakenne sisältää yhden osa-kokonaisuussuhteen kuvauksen ja sitä vastaavat ilmentymät. Komponenttitietokannalla tarkoitetaan kaikista tietokannassa olevista PSE-elementeistä muodostuvaa joukkoa.

PSE-rakenne esitetään parina (Ext,N), jossa Ext on ekstensionaalinen ja N intensionaalinen taso. Ekstensionaalinen taso on joukko, jonka alkioiden rakenne muodostetaan atomisista arvoista, olio-identiteetillä varustetuista tuple-rakenteista ja joukoista. Atomisilla arvoilla ei ole rakennetta. Ne ovat joko merkkijonoja tai lukuja. Nimiöidyllä tuplella on muoto oid<x_1,x_2,…x_n>, jossa x_1-x_n ovat alkioita ja *oid* on tuplelle generoitu olioidentiteetti. Joukko esitetään tyyliin {x_1,x_2,…x_n} missä x_1-x_n ovat joukon alkioita. Joukko- ja tuplekonstruktorit tulkitaan samoin, kuin edellä esitetyssä O^2-oliomallissakin.

Jokainen osa ja kokonaisuus on ekstensionaalisella tasolla kuvattu oliona. Niillä on täten yksikäsitteinen olioidentiteetti. Oliot esitetään tuple-rakenteen avulla. Vasemmalta oikealle mentäessä oliota esittävän tuplen alkiona on ensin attribuuttien arvot ja sitten olion mahdolliset komponentit. Jokaisella oliolla pitää olla ainakin yksi attribuutti, jotta indeksointimekanismia pystytään soveltamaan. Komponentit esitetään joukon sisällä kokonaisuuden tuple-rakenteen alkioina. Joukko antaa mahdollisuuden useamman samassa roolissa olevan komponentin esittämiseen.

PSE-esityksessä intensionaalinen taso esitetään binäärirelaationa, eli se sisältää joukon järjestettyjä pareja. Jokaisen parin ensimmäisenä elementtinä on joko olion tyypin tai attribuutin nimi ja toisena elementtinä indeksi, joka liittyy kyseiseen nimeen. Intensionaalisen tason esitystä N voidaan kutsua myös nimeämisrelaatioksi, joka kuvaa joukon indeksejä joukolle, joka muodostuu attribuuttien tai olioiden tyyppien nimistä. Binäärirelaatiossa tietty indeksi liitetään yhteen nimeen, mutta tiettyä nimeä voi vastata useita indeksejä.

Havainnollistan formalismin esitystapaa Junkkarin [2001] esittämällä polkupyöräesimerkillä. Käytän esimerkkiä myös myöhemmin kyselykielen toteutuksen, ja kyselyjen tekemisen yhteydessä. Kuva 2. esittää mallinnettavaa, tricyle-tyyppistä polkupyörää.

Kuva 2: Tricyclen komponentit

Kuvasta 2. nähdään, tricycle:n rakenne. *Tricycle* voidaan jakaa rakenteensa perusteella karkeasti neljään osaan. *Steering* muodostaa yhden erillisen kokonaisuuden. *Tricycle*:n keskiosassa on *saddle*. Takaosa koostuu kahdesta *wheel*:istä sekä niiden välillä olevasta axle:sta. *Frame* pitää *saddle*:a ja *rear*:ia toisissaan kiinni. On huomion arvoista, että *rear* ja *steering* ovat erityyppisessä osa-kokonaisuussuhteessa aikaisemmin esitetyn erottelun [Winston *et al.*, 1987] mukaan. *Rear* ja *tricycle* on eroteltu mieluummin osa / osuus massasta – erottelulla, sillä *rear* on tavallaan osuus koko *tricycle*:stä. Sen sijaan *steering* on mieluummin jäsennetty komponentti / kokonaisuuden muodostava objekti erottelun mukaisesti. Tämä johtuu siitä, että *steering*:illä on toiminnallinen rooli kolmipyörässä. Ei ole mitään sääntöä, joka pakottaisi ajattelemaan *rear*:in muodostuvan kahdesta *wheel*:istä ja axle:sta. Myös *steering* olisi voitu jäsentää jollakin muulla tavalla. PSE-esitystavalla voidaan mallintaa suhteet tarkoituksenmukaisiin tulkintoihin perustuen. Tällöin tulee kuitenkin ottaa huomioon, että jaottelussa osien ja kokonaisuuksien suhteiden kesken säilyy transitiivinen tulkinta.

Seuraavaksi havainnollistan kuvan 2. *tricycle*:n osiin jakamista Junkkarin [2001] esittämällä NF^2-relaatioille tyypillisellä visualisoinnilla. Sillä on havainnollista esittää sekä intensionaalinen että ekstensionaalinen taso. Kaavio muodostuu hierarkkisesta esittämistavasta, jossa kullakin tasolla on ensin mallinnettavan rakenteen nimi, jonka jälkeen välittömästi alempana on vasemmalta oikealle mentäessä sen attribuuttien nimet ja näiden jälkeen sen mahdolliset komponentit. Attribuuttien ja komponenttien nimien alapuolella on niitä vastaava ekstensionaalinen taso. NF^2-relaatioiden esitystavasta taulukkoesitys poikkeaa siinä, että NF^2-mallin sisäkkäiset relaatiot on taulukkoesityksessä korvattu olioilla / oliojoukoilla, joihin liittyy olioidentiteetti. Lisäksi samannimisiä

oliotyyppejä on mahdollista esittää samassa osa-kokonaisuusrakenteessa, mikä ei puolestaan ole mahdollista NF^2-relaatioiden yhteydessä. Tilanpuutteen takia joidenkin attribuuttien nimiä on lyhennetty. Attribuuttinimet *Diam*, *W*, *H*, *L*, *B* ja *R_Type* tarkoittavat vastaavassa järjestyksessä attribuutteja *Diameter*, *Weigth*, *Heigth*, *Length*, *Breadth* ja *Rim_type*.

TRICYCLE																										
oid	Price	FRAME			SADDLE		STEERING											REAR								
		oid	Frame-No	Material	oid	Pad	oid	H.	FRONT AXLE		HANDLEBAR		PEDALS		WHEEL			oid	B.	REAR AXLE			WHEEL		
									oid	L.	oid	B.	oid	Diam.	oid	Diam.	R_type			oid	Diam.	L.	oid	Diam.	R_type
o12	100	o9	4566545	steel	o10	plastic	o11	17	o5	4	o6	13	o7	5	o8	8	united	o4	13	o1	0.5	12	o2	6	united
																							o3	6	united

Kuva 3. Kolmipyörän intensionaalisen ja ekstensionaalisen tason sisältävä taulukkoesitys

Kuvassa 3. on selvyyden vuoksi harmaalla värillä sävytetty intensionaalinen taso ja valkoisella ekstensionaalinen taso. Esimerkkikaaviossa on siis vain yksi *tricycle*:n ilmentymä. Jos ilmentymiä olisi useampia, ne olisivat valkoisessa osassa omilla riveillään. Ekstensionaalisella tasolla vasemmalta oikealle mentäessä kuvataan ensimmäiseksi *tricycle*-olion identiteetti (oid), jonka arvona on esimerkissä o12. Tämän jälkeen tulevat *tricycle*:n attribuutit, joita esimerkin tapauksessa on vain yksi eli *Price*. Attribuutin arvona on 100 ekstensionaalisella tasolla. Tämän jälkeen kuvataan *tricycle*:n komponentit, *frame*, *saddle*, *steering* ja *rear*. Kaikki *tricycle*:n komponentit, jotka ovat samalla myös kompositioita, esitetään samalla periaatteella kuin *tricycle* itsekin. Komponentit, jotka eivät ole kompositioita, esitetään muutoin samalla tavalla kuin kompositiot, mutta luonnollisesti ilman komponentteja. Tällaisia perustyyppejä esimerkkikaaviossa ovat *front axle*,*handlebar*,*pedals*,*wheel*,*rear axle* ja *wheel*.

Visualisoinnissa intensionaalinen ja ekstensionaalinen taso on sidottu toisiinsa selkeällä tavalla. Intensionaalisen tason miltä tahansa kohdalta voidaan siirtyä tarkastelemaan sitä vastaavia ekstensionaalisia arvoja. Myös toiseen suuntaan tarkastelu on mahdollista. Visualisoinnista näkee esimerkiksi sen, että arvoa 17 vastaa *tricycle*:n komponentin *steering*:in *heigth*-attribuutti intensionaalisella tasolla.

PSE-esitystavassa taulukkoesityksen harmaa alue esitetään siis binäärirelaationa. Taulukon intensionaalisesta osasta huomataan sen olevan muodostettu tavalla, joka luo mahdollisuuden liittää tietty indeksi sen jokaiselle attribuutille ja komponentille. Ylimpään kokonaisuuteen, eli *tricycle*:een, voidaan liittää indeksi <1>. Kun mennään osa-kokonaisuushierarkiatasolta yksi taso alaspäin, indeksin pituutta kasvatetaan yhdellä, eli indeksi on tällöin muotoa <1,X>, missä X on jokin kokonaisluku. Vasemmalta oikealle mentäessä X:n arvoa kasvate-

taan. Kolmipyörän tapauksessa vasemmalta oikealle mentäessä sen ensimmäisenä olevassa attribuuttissa (*Price*) arvotetaan yhdeksi ja seuraavaksi tulevassa komponentissa (*frame*) X arvotetaan kahdeksi. Toisin sanoen *price*-attribuuttia vastaa indeksi <1,1> ja *frame*:a <1,2>. On huomionarvoista, että olioidentiteettiin ei liitetä indeksiä lainkaan, koska siihen ei liity osa-kokonaisuussuhteissa intensionaalisella tasolla omaa semanttista tulkintaa. Samalla periaatteella *front axle* –komponentin *length*-attribuuttiin liitetään indeksi <1,4,2,1>, sillä indeksi <1,4> liittyy *steering*:iin ja indeksi <1,4,2> *steering*:in komponenttiin *front axle*:en, ja lopulta *front axle*:n ensimmäiseen ja ainoaan attribuuttiin *length* indeksi <1,4,2,1>. Indeksit kuvaavat sisäkkäisiä hierarkiatasoja, eli attribuuttien ja komponenttien sijaintia oliotyypissä / oliossa. Taulukkoesityksessä olevaan ekstensionaaliseen tasoon ei kuitenkaan pystytä indeksejä hyödyntämään, sillä taso on esitetty tasaisena rakenteena. Kaaviossa tasot on sidottu toisiinsa vain visuaalisesti.

PSE-esitystavassa tieto on kuitenkin esitetty tavalla, jossa intensionaalisella tasolla esiintyvät indeksit on mahdollista liittää myös olioille ja attribuuttien arvoille. Ekstensionaalinen taso on esitetty PSE-esityksessä aiemmin esitetyllä kolmella rakenteella. Tuple-rakenteen ansiosta ekstensionaalinen taso esitetään systemaattisesti siten, että se mahdollistaa indeksien soveltamisen. Tuple-esityksessä jokaisella elementillä on positio, johon indeksissä olevalla kokonaisluvulla voidaan viitata. Tämän takia jokainen olio on kuvattu tuple-rakenteella PSE-esityksessä. Tuple-rakenteen alkioina ovat attribuutin arvot ja oliot. Kuvassa 3. *tricycle*:n rear-komponentissa *wheel*-oliotyyppi on esitetty vain kerran. Kuitenkin kuvassa *wheel*:iä vastaa kaksi ilmentymää ekstensionaalisella tasolla. *Wheel*-oliot esitetään siinä allekkain. Toisen olioidentiteetti on o2 ja toisen o3. Tämä ilmentää sitä, että *rear*-osassa *wheel*-käsitettä vastaa joukko *wheel*-olioita, joiden järjestyksellä ei ole väliä. Tässä tapauksessa joukossa on kaksi oliota. Niistä ei voi päätellä kumpi on vasemmanpuoleinen ja kumpi oikeanpuoleinen *wheel*-olio. Tällä tavalla pystytään kuvaamaan *jäsen/kokoelma* -suhdetta. PSE-esitystavassa tämänkaltainen tilanne on ratkaistu siten, että jokainen olio on joukon sisällä. Oliojoukkoon liittyy aina vain yksi indeksi. Jos joukossa on useampi olio, tämä merkitsee sitä, että kaikkiin joukossa oleviin olioihin liittyy sama indeksi. Alla on Junkkarin [2001] antama PSE-esitys yllä olevasta *tricycle*-esimerkistä. Ensin kuvataan ekstensionaalinen taso:

{o12<100, {o9<4566545, steel>}, {o10<plastic>}, {o11{17,{o5<4>},{o6<13>}, {o7<5>}, {o8<8,united>}>}, {o4<13,{o1<0.5,12>}, {o2<6,united>, o3<6,united>}>}>}

Yllä oleva ekstensionaalinen taso on formaali esitystapa kuvassa 3 esitetylle ekstensionaalisen tason visualisoinnille. *Tricycle*-olio ilmaistaan uloimpien aaltosulkujen sisällä. Aaltosulut tarkoittavat joukkoa ja kuvattu ilmentymä on joukon ainoa alkio. Ilmentymä ilmaistaan kulmasuluilla, eli tuplerakenteena. Tuple varustetaan olioidentiteetillä. Koko *tricycle*:n olioidentiteetti on siis o12, ja sen ainoan attribuutin arvona on 100. Kyseessä oleva arvo esiintyy tuplen ensimmäisessä positiossa. Tämän jälkeen tulevat alkiot ovat joukkoja, joiden sisällä *tricycle*:n komponentit esitetään tuple-muotoisina. *Tricycle*:n komponentit konstruoidaan samalla periaatteella. *Tricycle*:ä kuvaavan tuplen viimeisenä alkiona on joukko, jonka sisällä on kaksi oliota, joiden olioidentiteettinä ovat *o2* ja *o3*. Ne siis ovat *rear*-komponentin *wheel*-olioita. *Tricycle*-olioon, liitetään indeksi <1>. Sen ensimmäiseen alkioon, eli *Price*-attribuutin arvoon 100, liittyy indeksi <1,1>. Vastaavasti attribuuttiarvoon *steel* liittyy indeksi <1,2,2>.

Alla on *tricycle*:n intensionaalisen tason formaali esitys. Binäärirelaatiosta voidaan selvittää esimerkiksi se, mikä on attribuuttiarvoa *steel* vastaava käsite intensionaalisella tasolla. Koska ekstensionaalisella tasolla attribuuttiarvoon *steel* voidaan liittää indeksi <1,2,2> ja intensionaalisella tasolla attribuuttiin *material* liitetään sama indeksi, on *steel* attribuutin *material* arvo.

{<TRICYCLE, <1>>, <PRICE, <1,1>>, <FRAME, <1,2>>,
<FRAME_NO,<1,2,1>>, <MATERIAL, <1,2,2>>, <SADDLE, <1,3>>, <PAD,
<1,3,1>>, <STEERING, <1,4>>, <HEIGTH, <1,4,1>>, <FRONT_AXLE, <1,4,2>>,
<LENGTH,<1,4,2,1>>, <HANDLEBAR, <1,4,3>>, <BREADTH,<1,4,3,1>>,
<PEDALS, <1,4,4>>, <DIAMETER, <1,4,4,1>>, <WHEEL, <1,4,5>>,
<DIAMETER, <1,4,5,1>>, <RIM_TYPE, <1,4,5,2>>, <REAR, <1,5>>,
<BREADTH,<1,5,1>>, <REAR_AXLE, <1,5,2>>, <DIAMETER,<1,5,2,1>>,
<LENGTH,<1,5,2,2>, <WHEEL, <1,5,3>>, <DIAMETER, <1,5,3,1>>,
<RIM_TYPE, <1,5,3,2>>}

Kuvan 3. PSE-esitys muodostuu siis kahdesta edellä annetusta formaalista esitystavasta. Usein osa-kokonaisuussuhteessa ilmentymiä on enemmän kuin yksi.

Voimme siis ekstensionaalisella tasolla esiintyvän arvon / rakenteen tulkita vastaavaksi intensionaalisen tason käsitteeksi ja päinvastoin. Lisäksi indeksejä analysoimalla pystymme selvittämään monenlaista informaatiota olioiden / oliotyyppien keskinäisistä suhteista [Junkkari, 2001].

5.2.1. Indekseihin perustuva analysointi PSE-esitystavassa

Indekseihin sisältyy runsaasti informaatiota osa-kokonaisuusrakenteesta, jos tieto on järjestetty PSE-esityksen edellyttämällä tavalla. Intensionaalisen ja ekstensionaalisen tason integroinnin lisäksi indeksien avulla voidaan suoraan analysoida rakenteellisuutta ja suhteita rakenteiden välillä. Tässä tutkielmassa indeksien analysointia esitellään suppeasti ja yleisellä tasolla. Formaalimpi indeksien analysointi on esitetty Junkkarin [2001] työssä.

Indeksien demonstroimisessa käytän seuraavaa esitystä: Esitän indeksit muodossa <x1,x2,x3,...,xn>, missä x1-xn ovat kokonaislukuja. Symbolit I1-In tarkoittavat mielivaltaisia indeksejä. Ne voivat siis myös olla tyhjiä indeksejä. Täten <I1,x1> tarkoittaa indeksiä, jonka alkuna on indeksi I1 ja viimeisenä alkiona x1. Indeksi <I1,x1,x2,I2,x3> tarkoittaa indeksiä, jossa indeksin I1 jälkeen esiintyvät alkiot x1 ja x2 peräkkäin. Tämän jälkeen tulee mielivaltainen indeksi I2, jonka jälkeen viimeisenä alkiona on x3. Jos siis x1 = 1, x2 = 2 ja x3 = 3, niin esimerkiksi indeksi <3,2,2,4,5,1,2,6,8,3> olisi eräs yllä olevan indeksirakenteen hyväksymä indeksi.

Indeksi I1 liittyy tyyppiin, jos PSE:stä löytyy muotoa <I1,x1> oleva indeksi. Tämä johtuu siitä, että kaikilla tyypeillä on oltava vähintään yksi attribuutti. Attribuutti-indeksi on puolestaan sellainen indeksi, jolle ei ole löydettävissä samasta PSE:stä samanalkuista, mutta pidempää indeksiä. Peruskomponentille kuvautuva indeksi I1 on sellainen, että kaikki muotoa <I1,x1> olevat indeksit ovat attribuutti-indeksejä. Tämä tarkoittaa siis sitä, että indeksiin kuvautuvalla tyypillä ei ole komponentteja. Tiettyä tyyppiä kuvaavan indeksin I1 komposiitti-indeksit ovat indeksejä, jotka ovat alusta samanlaisia, mutta lyhempiä kuin <I1>. Esimerkiksi tyyppi-indeksin <1,3,4,2> komposiitti-indeksit ovat <1,3,4,>, <1,3> ja <1>. Tietyn indeksin <I1> komponentti-indeksit ovat puolestaan sellaisia, jotka ovat alusta samanlaisia, mutta pidempiä ja lisäksi tyyppi-indeksejä. Esimerkiksi <1,2> indeksin komponentti-indeksejä ovat <1,2,4> ja <1,2,5>, mikäli molemmat indeksit liittyvät tyyppeihin. Komponentti- ja komposiitti-indeksit helpottavat myös transitiivisten yhteyksien selvittämistä.

5.3. Ohjelmointikielen valinta kyselykielen toteuttamiseen

Jotta PSE-esitystavan pohjalta voitaisiin ohjelmoida tietokoneelle toimiva kyselykieli, tarvitaan ohjelmointiparadigma, joka tukee esitystavan mukaista informaation esittämistä ja käsittelyä. Aikaisemmin totesin relaatiomallin, sen laajennoksen NF^2-mallin ja olio-orientoituneiden tietokantojen olevan riittämättömiä helppokäyttöisen, mutta samalla ilmaisuvoimaisen kyselykielen vaatimuksille.

Toteutuskielen tulee täyttää kaksi perusvaatimusta, jotta sen avulla PSE-esitystapa voitaisiin siirtää helposti käytäntöön. Ensiksikin sen on hyvä olla luonteeltaan deduktiivinen. Tämän johdosta pystytään tekemään korkean deklaratiivisen asteen omaava kyselykieli. Toiseksi kielen tulee tukea olio-orientoitunutta esitystapaa, joka on keskeisellä sijalla PSE-esitystavassa. Toisin sanoen paradigma tarjoaa automaattisen olioidentiteetin generoinnin ja metodien määrittelyn. Metodeja tarvitaan johdettujen attribuuttien mallintamisessa.

Näitä vaatimuksia silmälläpitäen kielen toteutusvälineeksi valittiin logiikkaohjelmointiparadigman ja olio-orientoituneisuuden paradigman piirteitä sisältävä hybridikieli Prolog++. Kieltä voidaan pitää deduktiivisena olio-orientoituneena ohjelmointiparadigmana [Moss, 1994]. Kielen valintaa tukevat myös sen käytöstä saadut hyvät kokemukset deduktiivisten oliotietokantoihin perustuvan kyselykielen prototyyppitoteutuksen yhteydessä [Niemi *et al.*, 2000]. Ennen kielen piirteiden esittelyä tarkastelen muutamia logiikkaohjelmointiparadigman ja Prologin peruskäsitteitä.

5.3.1. Logiikkaohjelmointi

Logiikkaohjelmointi perustuu nimensä mukaisesti logiikan sääntöihin. Logiikkaohjelmointi on joukko faktoja ja sääntöjä, jotka määrittelevät suhteita objektien välille. Logiikkaohjelmassa määritellään faktojen ja sääntöjen avulla suljettu maailma, joka määrittelee siitä vedettävissä olevien johtopäätösten joukon. Logiikkaohjelman ajaminen tarkoittaa ohjelman faktojen ja sääntöjen määräämän suljetun maailman pohjalta tehtyä päättelyä [Sterling and Shapiro, 1986]. Logiikkaohjelman ajaminen suoritetaan tekemällä kyselyjä ohjelmalle. Kielen avulla on mahdollista toteuttaa kaikki relaatiotietokantojen esittämiseen ja käsittelyyn liittyvät piirteet [Niemi and Järvelin, 1991; Paton *et al.*, 1996]. Lisäksi sillä pystytään määrittelemään rekursiivisia rakenteita, minkä johdosta transitiivisten suhteiden käsittely mahdollistuu. Logiikkaohjelmassa kaikki informaatio esitetään vakioista, muuttujista ja funktoreista koostuvina termeinä. Vakiot ovat termejä, joita ei voi jakaa pienempiin osiin. Muuttujalla esitetään mielivaltaista argumentin arvoa. Tässä esityksessä pitäydyn Prologin tavassa merkitä muuttujia isoilla ja vakioita pienillä kirjaimilla. Logiikkaohjelman faktat ilmaisevat olion ominaisuuksia tai olioiden välillä vallitsevia suhteita. Tarkastellaan seuraavaa faktajoukkoa.

```
putkimies(mikko).
toimitusjohtaja(pekka).
asuu(mikko,tampere).
asuu(pekka ,rovaniemi).
```

Kaksi ensimmäistä faktaa ovat yksipaikkaisia predikaatteja. Faktojen nimet kirjoitetaan pienillä kirjaimilla. Logiikkaohjelmoija antaa faktoille tulkintoja. Ensimmäinen fakta voidaan tulkitaan siten, että *mikko* on putkimies. Predikaatin nimenä on putkimies ja argumenttina *mikko*. Predikaatti voidaan tulkita myös siten, että *putkimies* on annettu ominaisuus ja *mikko* annettu objekti, jolla on kyseinen ominaisuus. Kaksi jälkimmäistä faktaa ovat kaksipaikkaisia predikaatteja, ja ne voidaan tulkita kahden objektin väliseksi relaatioksi. Jälkimmäiset kaksi faktaa voidaan tulkitaan siten, että ensimmäinen argumentti kuvaa henkilöä, joka asuu toisena argumenttina annetun objektin ilmaisemassa paikassa. Esimerkiksi viimeinen fakta voitaisiin kirjoittaa myös muodossa asuu(rovaniemi,pekka), mutta tulkinta pysyisi samana, jos logiikkaohjelmoija päättäisi, että ensimmäinen argumentti kuvaa asuinpaikkaa ja toinen argumentti asujaa. Tulkinta on siis ohjelmoijan vastuulla. Jos logiikkaohjelma muodostuu yllä olevista faktoista, voimme tehdä kyselyjä kyseessä olevaan logiikkaohjelmaan perustuen. Kysely ”putkimies(mikko)?” tuottaa vastaukseksi yes indikoimaan, että se on ohjelmasta vedettävissä oleva johtopäätös. Sen sijaan kysely ”putkimies(pekka)?” tuottaa vastaukseksi no. Myös muuttujaa voi käyttää kyselyissä. Esimerkiksi kysely ”asuu(pekka,X)?” voidaan tulkita kyselyksi, missä *pekka* asuu. Vastaukseksi kyselyyn saadaan tällöin ”X = rovaniemi”. Kysely ”asuu(X,Y)?” puolestaan antaisi kaksi vastausta,

X = mikko
Y = tampere,

ja

X = pekka
Y = rovaniemi.

Kyselyn prosessointi arvottaa siis muuttujat kaikilla mahdollisilla tavoilla, joilla kysely on ohjelmasta vedettävissä oleva johtopäätös.

Säännöt ovat logiikkaohjelmoinnissa tapa määrittää yhteyksiä objektien välille. Säännöillä on seuraava muoto: X <- Y1,Y2..Yn. X:ää kutsutaan *Säännön pääksi* ja rungon muodostavat Yi:t, joita kutsutaan tavoitteiksi. Tavoite itsessään voi olla fakta tai sääntö. Seuraavassa annan esimerkin isoisän määrittelevästä säännöstä.

isoisä(X,Y) <- vanhempi(Z,Y),isä(X,Z).

Sääntö tulkitaan luonnollisen kielen ilmaisuna seuraavasti: jotta X on Y:n isoisä, täytyy löytyä henkilö Z, joka on Y:n vanhempi ja jonka isä X on. Sääntö edellyttää, että ohjelmasta löytyvät rungon tavoitteita vastaavat säännöt tai faktat. Sääntöjen soveltamisessa tarvitaan jaetun muuttujan ideaa. Tämä tarkoittaa sitä, että säännön päässä ja rungon tavoitteissa oleva saman niminen muuttuja tarkoittaa samaa objektia kaikkialla säännössä. Säännön rungossa olevilla pilkuilla on sama merkitys kuin konjunktiolla logiikassa.

Rekursiivista määrittelyä demonstroidaan seuraavalla esimerkillä. Luonnollinen luku voidaan määritellä esimerkiksi seuraavalla, rekursiivisella tavalla:

```
lluku(0).
lluku(n(X)) <- lluku(X).
```

Tällöin esimerkiksi n(0) ja n(n(n(0))) ovat luonnollisia lukuja. Sääntö siis määrittelee yksikäsitteisesti, mitkä termit ovat luonnollisia lukuja. Rekursion lopetusehtona on fakta lluku(0).

Listat ovat logiikkaohjelmoinnissa tehokas tiedon esittämistapa. Listojen rakenne on binäärinen. Listaesitys voidaan määritellä logiikkaohjelmalla seuraavasti.

```
lista([]).
lista([X|Y]) <- lista([Y]).
```

Listan käsittelyssä on kolme tärkeää käsitettä. Tyhjää listaa merkitään hakasuluilla tyyliin []. Yllä olevassa esimerkissä listan pää on X ja häntä Y. Esimerkki tarkoittaa luonnollisella kielellä sitä, että tyhjä lista on laillinen lista ja sen lisäksi sellainen lista, joka voidaan jakaa päähän ja häntään on laillinen, jos häntä on laillinen lista. Määritelmä on siis rekursiivinen. Esimerkiksi rakenne [a,b,c,d,e] on laillinen lista, jonka pää on a ja häntä [b,c,d,e]. Listan [e] pää on e ja häntä [].

5.3.2. Prolog

Logiikkaohjelmointi on yleiskäsite, eikä siis itsessään varsinainen ohjelmointikieli. Se on mieluummin yleinen kehys, joka määrittelee logiikkaohjelmointiparadigmassa käsiteltävät primitiivit. Prolog on prosessointitapa, jossa määritellään järjestys sekä ohjelmassa oleville lauseille että lauseen rungon tavoitteille. Prolog on konkreettinen ajomalli logiikkaohjelman ajamiseksi. Logiikkaohjelmaa, joka ajetaan Prologin prosessointimekanismilla, kutsutaan puhtaaksi Pro-

logiksi. Se on eräs tapa toteuttaa logiikkaohjelman ajaminen peräkkäiskäsittelyyn orientoituneella tietokoneella. Prolog prosessoi kyselyjen ratkaisemisen yhteydessä lauseita niiden esittämisjärjestyksessä ja rungon tavoitteita vasemmalta oikealle. Prologilla ohjelmoitaessa ei siis päästä tästä syystä täysin eroon proseduraalisesta ajattelutavasta, vaan ohjelmoijan on otettava tämä ominaisuus huomioon. Joskus lauseiden ja tavoitteiden järjestys voi muuttaa ohjelman merkitystä. Prolog simuloi epädeterminististä valintaa *samaistamisalgoritmin* ja *peruutuksen* avulla [Sterling and Shapiro, 1986]. Lisäksi lauseiden suoritusjärjestykseen voi vaikuttaa leikkaukseksi kutsutun menetelmän avulla.

5.3.3. Prolog++

Prolog++ on kieli, joka on rakennettu normaalin Prologin päälle lisäämällä siihen olio-ohjelmoinnin piirteitä. Se tukee esimerkiksi luokan, olion, luokan metodin ja periytymisen käsitteitä. Myös olio-ohjelmoinnista tutut suojausmääreet *private* ja *public*, voidaan antaa metodeille ja attribuuteille. Kieli tukee myös poikkeuksien käsittelyä [Moss, 1994]. Kieli yhdistää siis logiikkaohjelmoinnille tyypillisen deklaratiivisuuden ja olio-orientoituneisuudelle tyypillisen mallintamisvoiman. Siksi sen on valittu tutkielmassa kehitetyn kyselykielen toteutuskieleksi.

Prolog++:ssa luokka määritellään esittelemällä luokan nimi ja siihen liittyvät attribuutit ja metodit. Attribuutit ovat loogisia muuttujia, kuten Prologissa yleensä. Metodit ovat puolestaan normaaleja Prolog–predikaatteja, eli ne voivat olla faktoja ja sääntöjä. Jos luokan komponentteina on muita luokkia, ne määritellään ilmaisulla *parts* osa1, osa2,…,osaN, jossa osat 1-N ovat muualla ohjelmassa määriteltyjä luokkia. Kun osia sisältävä luokan ilmentymä luodaan, myös sen osat luodaan samalla kertaa. Tämän ominaisuuden johdosta kielellä voidaan tehdä luonteeltaan vain poissulkevia komponentteja. Luokan määrittämisen demonstroimisessa käytän aiemmin esitettyä kolmipyöräesimerkkiä. Käytän sitä myös myöhemmin Prolog-perustaista PSE-esitystapaa esiteltäessä. Esimerkki poikkeaa PSE-esimerkistä vain siinä, että tässä esimerkissä mukana on myös metodi *weigth*, joka palauttaa osiensa painon summan.

```
class tricycle.
  public instance attribute price.
  parts frame,saddle,steering,rear.
      weigth(X) :- frame#1<-weigth(X1),saddle#1<-weigth(X2),steering#1<-
  weigth(X3),rear#1<-weigth(X4),X is X1+X2+X3+X4.
end tricycle.
```

Luokan määrittäminen aloitetaan *class* –sanalla, jota seuraa luokan nimi. Määrittely päätetään *end* -sanaan, jota seuraa uudelleen luokan nimi. Luokalla on julkisena attribuuttinaan *price* ja osina *frame, saddle, steering* ja *rear*. *Weight* on luokan metodi, joka hakee osiin liittyvät painot ja laskee ne yhteen.

Esimerkin *tricycle*-luokan mukainen ilmentymä voidaan luoda seuraavasti: tricycle <- create(X). Nuoli *tricycleen* päin tarkoittaa, että *tricycle*-luokalle välitetään viesti create(X). *Create* on systeemimetodi, joka on käytettävissä kaikilla luokilla. Se luo luokkaan kuuluvan ilmentymän ja samaistaa X-muuttujan luodun olion oliotunnisteen kanssa. Luokan ilmentymän tunniste esitetään terminä (X|Y), missä X on luokan nimi ja Y on olioidentiteetti. Yllä oleva luontilause voisi tuottaa esimerkiksi oliotunnisteen (tricycle|543251). Oliolle voidaan sen luonnin jälkeen lähettää viestejä samaan tapaan kuin luokallekin. Viestien lähettämistä voidaan käyttää moneen tarkoitukseen. Esimerkiksi hinta voidaan kysyä yllä olevan esimerkin tapauksessa tavoitteella (tricycle|543251) <- price(X), jolloin X arvottuu olion *price*-attribuutin sen hetkisellä arvolla. Metodin palauttama arvo ilmaistaan samalla tavalla kuin attribuuttien yhteydessä. Aiemmin luodun *tricycle*-olion paino saadaan tavoitteella (tricycle | 543251) <- weigth(X). Tavoitteessa olevan muuttujan alustaminen merkitsee arvon palautusta, kun taas proseduraalisten olio-ohjelmointikielissä palautusarvo yleensä sijoitetaan johonkin muuttujaan. Metodin parametria käytetään myös normaalien parametrien tavoin välittämään metodille jokin arvo käsiteltäväksi. Tällöin parametriksi annetaan jokin luku tai muu vakio. Parametrejä voi olla metodilla enemmänkin kuin yksi kappale.

5.4. Prolog++ -toteutustapa PSE-esitystavalle

PSE-esitystavan Prolog++ toteutus edellyttää, että sen sisältämä informaatio esitetään Prolog-termeillä. PSE-esitystavassa ei käsitellä dynaamista muistin hallintaa. Luodut oliot sijaitsevat tietokoneen muistissa. Toteutuksessa tulee siis ottaa huomioon intensionaalisen ja ekstensionaalisen tason Prolog++:n mukaisen esitystavan integrointi PSE-esitystavan kanssa.

Prolog-termien mallintamisessa sovellan Niemen ja Järvelinin [1991] ehdottamia Prolog-pohjaisia konstruktoreita. Toteutuksessa käytän tuple-, joukko- ja kuvauskonstruktoreita (*map*). Tuple-rakenne esitetään terminä t(X1,X2,…,Xn), missä x1-xn ovat mitä tahansa laillisia Prolog-termejä. Tuple-esitetään siis yhdistettynä Prolog-terminä, jonka funktorin nimenä on *t* ja jonka komponenttien määrä voi vaihdella yhdestä n:ään. Esitystavan käsittelyn yhteydessä voidaan hyödyntää systeemipredikaatteja, jotka on tarkoitettu yhdistettyjen Prolog-termien analysointiin.

Kuvauskontruktori on tarkoitettu ilmaisemaan yhteydet kahden joukon elementtien välillä. Kuvausesitystavan matemaattisena taustakäsitteenä on binäärirelaatio. Kuvauskonstruktori on siis joukko, jonka alkioina on map-pareja. Map-pari esitetään puolestaan yhditettynä terminä map(X,Y), missä termi X kuvaa toisen joukon alkion ja termi Y toisen. On huomionarvoista, että termit X ja Y voivat olla myös monimutkaisen rakenteen sisältäviä Prolog-termejä.

Tieto esitetään joukkona, jos on tarve mallintaa rakenteellisesti homogeenisiä alkioita sisältävä ryhmä, jonka alkioiden järjestyksellä ei ole merkitystä. Valitettavasti standardi Prolog ei tarjoa joukkoa esittävää tietorakennetta [Liu, 1999]. Tämän vuoksi joukkojenkin esittämiseen käytän muun muuassa Niemen ja Järvelinin [1991] ehdottamaa esitystapaa. Siinä joukot esitetään listoina. Tätä valintaa tukee muun muassa monet listoille määritellyt systeemipredikaatit. Esimerkiksi predikaattia member(X,L) voidaan soveltaa tutkimaan, kuuluuko alkio X joukkoon L.

Edellä mainitsemieni kolmen konstruktorin avulla voidaan PSE-esitystapa esittää Prolog-perustaisesti. Kuten yllä todettiin, esitystapa joudutaan lisäksi integroimaan Prolog++:n mukaisten olioiden kanssa. PSE-formalismin [2001] tapa esittää oliot joukko-opillisesti ei ole riittävä. Tämä johtuu siitä, että olioille täytyy mallintaa metodeja, jotka mahdollistavat osa-kokonaisuussuhteelle tyypillisen periytymismekanismin toteuttamisen. Tämän vuoksi olioiden tulee olla dynaamisina tietokoneen muistissa ajon aikana. Olioita haetaan muistista oliotunnisteiden avulla. Tieto oliotunnisteista tulee siis olla olioita esittävissä termeissä. Nämä olion, ja sitä vastaavan termin integroinnin seikat tulee ottaa huomioon luokkaa määriteltäessä.

5.4.1. Esitystavan edellyttämän olion ja luokan piirteet

Intensionaalisen ja ekstensionaalisen tasojen toteuttamisessa tulee ottaa huomioon tasojen integroinnin yhteydessä esiintyvät tekniset vaatimukset. Intensionaalinen taso esitetään Prolog-terminä sen jälkeen, kun Prolog++ luokka on ensin määritelty sille. Esimerkkinä käytän aiemmin esittelemääni *tricycle*-esimerkkiä. Erona Junkkarin esittämään *tricycle*:en on kuitenkin se, että kaikissa sen komponenttityypeissä on attribuuttina *weigth. Weigth* on johdettu attribuutti kaikissa *tricycle*:n kompositiotyypeissä, siis myös itse *tricycle*-oliotyypissä. Perusoliotyypissä *weigth* on normaali attribuutti, jota ei johdeta mistään. Intensionaalisessa tasossa *weigth* esitetään kuitenkin myös komposiittienkin kohdalla normaalina attribuuttina, johon liittyy attribuutti-indeksi. Alla on intensionaalisen tason Prolog-perustainen toteutus kuvan 2. *tricycle*:stä.

pse([map(tricycle,t(1)),map(price,t(1,1)),map(weigth,t(1,2)),map(frame,t(1,3)),map(saddle,t(1,4)),map(steering,t(1,5)),map(rear,t(1,6)),map(frame_no,t(1,3,1)),map(material,t(1,3,2)),map(weigth,t(1,3,3)),map(pad,t(1,4,1)),map(weigth,t(1,4,2)),map(h,t(1,5,1)),map(weigth,t(1,5,2)),map(front_axle,t(1,5,3)),map(handlebars,t(1,5,4)),map(pedals,t(1,5,5)),map(wheel,t(1,5,6)),map(b,t(1,6,1)),map(weigth,t(1,6,2)),map(rear_axle,t(1,6,3)),map(wheel,t(1,6,4)),map(l,t(1,5,3,1)),map(weigth,t(1,5,3,2)),map(b,t(1,5,4,1)),map(weigth,t(1,5,4,2)),map(diam,t(1,5,5,1)),map(weigth,t(1,5,5,2)),map(diam,t(1,5,6,1)),map(r_type,t(1,5,6,2)),map(weigth,t(1,5,6,3)),map(b,t(1,6,1)),map(weigth,t(1,6,2)),map(diam,t(1,6,3,1)),map(l,t(1,6,3,2)),map(weigth,t(1,6,3,3)),map(wheel,t(1,6,4)),map(diam,t(1,6,4,1)),map(r_type,t(1,6,4,2)),map(weigth,t(1,6,4,3))]).

Intensionaalista tasoa esittävän Prolog-termin funktorina on atomi *pse*. Intensionaalisen tason binäärirelaation esitän kuvaustyyppisenä rakenteena. Siinä *map*-parin ensimmäisenä elementtinä on attribuutin tai olion tyypin nimi ja toisena siihen liittyvä indeksi. Esitän indeksin tuple-konstruktorina. Esimerkkinä on vain *tricycle*:n intensionaalisen tason esitys. Jos intensionaalisella tasolla olisi mallinnettava muitakin osa-kokonaisuussuhteita, niin ne olisivat omina listoinaan *pse*-termin elementteinä.

Aikaisempi *tricycle*:ä kuvaava Prolog++:lla kuvattu luokka ei ole riittävä ekstensionaalisen tason esittämiseen. Luokaa muodostettaessa täytyy kyetä luomaan PSE-formalismin ekstensionaalinen esitystapa. Toisin sanoen olion täytyy pystyä muodostamaan itsestään PSE-esitystavan mukainen ekstensionaalinen esitys. Prolog-esityksen tulee olla lisäksi sellainen, että sitä voidaan hyödyntää Prolog++:n yhteydessä. Tämä tarkoittaa sitä, että ekstensionaalisen tason termiesityksestä tulee pystyä siirtymään muistissa olevaan olioon.

Jotta olio pystyy muodostamaan Prolog-esityksen itsestään, täytyy oliota vastaavassa luokassa määritellä metodi, joka koostaa olion osista oikeanlaisen esityksen. Määrittelen metodin siten, että kaikki oliot tulostavat itsestään tuple-rakenteen. Termin nimenä on merkkijono, jonka alku muodostuu t-kirjaimesta. Tämä vain siksi, että t-kirjain osoittaa kyseessä olevan tuple-konstruktorin. Välittömästi t-kirjaimen jälkeen merkkijonossa on oliota vastaavan luokan nimi. Tämän jälkeen merkkijonossa on olioidentiteetti. Jos kyseessä on perusolio, olio muodostaa itsestään tuple-rakenteen, jonka sisältönä ovat olioiden attribuuttien arvot. Jos sen sijaan on kyse kompositiosta, tuple-esityksen sisällä on attribuuttien jälkeen alkioina sen komponenttien esitys tuple-rakenteena. Kompositioiden kohdalla rakenne muodostetaan siis rekursiivisesti. Rakenteen rekursiivi-

nen muodostaminen määritellään siten, että komposition muodostaessa itseänsä se ensin muodostaa komponenttinsa. Nämä puolestaan antavat tulostuskäskyn omille komponenteilleen ja niin edelleen, ellei kyseessä ole perusolio. Perusolion kohdalla muodostaminen loppuu, sillä perusoliot koostuvat vain omien attribuuttiensa arvoista. Tällä tavalla ylimmän tason olio koostaa termin koko kokonaisuudesta. Alla esitän *print*-metodin tähän tarkoitukseen. Metodi pitää lisätä aikaisemmin Prolog++:n yhteydessä esittämääni *tricycle*-luokan määritykseen. Muuten ekstensionaalisen tason PSE-termin muodostaminen ajon aikana ei ole mahdollista.

```
print(X):-(Class | No) = self,name(Class,Casc), name(No,Noasc), ap-
pend(Casc,Noasc,Result1), name(t,Te), append(Te,Result1,Result),
name(Oid,Result), frame#1 <- print(X1),saddle#1 <- print(X2), steering#1 <-
print(X3),rear#1 <- print(X4), weigth(Weigth), X =.. [Oid,@price,Weigth,
[X1],[X2],[X3],[X4]].
```

Print-metodi arvottaa X muuttujan oliosta saatavalla ekstensionaalisen tason esityksellä. Kun olio luodaan, oliota luova metodi luo ensin olion halutuilla attribuuttien alkuarvoilla. Tämän jälkeen print-metodilla muodostetaan olion ekstensionaalinen esitystapa, joka liitetään Prolog-koodiin. Tämän jälkeen oliota edustava termi on Prolog-koodissa mukana seuraavalla ajokerralla. Liittäminen tapahtuu Prologin systeemipredikaatilla *assert*. Aina kun uusi olio luodaan, Prolog-koodiin generoidaan uusi termi esittämään olion tilaa. Esitystavan taustaoletuksena on siis se, että aina kun olion tila muuttuu, se poistaa vanhan PSE-esityksen koodista ja korvaa sen uudella. Tässä tutkielmassa en kuitenkaan käsittele päivityksiin liittyviä asioita, joten en ole toteuttanut prototyyppiin päivitysoperaatioita. Alla esitän *tricycle*:n ekstensionaalisen tason, joka syntyy print-metodin tuloksena.

```
ttricycle53607(100,25,[tframe53617(4566545,steel,4)], [tsaddle53627 (plastic,1)],
[tsteering53637(17,20,[tfront_axle53647(4,14)], [thandlebars53657(13,1)], [tped-
als53667(5,1)], [twheel53677(8,united,1)])], [trear53687(13,3,
[trear_axle53697(0.5,12,1)], [twheel53707(6,united,1),twheel53717(6,united,1)])])
```

5.4.2. Indeksointimekanismi toteutus Prologilla

Kun olio esitetään print-metodin muodostamalla termillä, voidaan eräällä toteuttamallani Prolog-termillä hakea rakenteesta arvo tai olio indeksin perusteella. Erällä toisella toteuttamallani Prolog-termillä puolestaan saadaan rakentees-

ta tiettyä arvoa tai oliota vastaava indeksi tai indeksit selville. Nämä kaksi Prolog-termiä mahdollistavat intensionaalisen ja ekstensionaalisen tason kaksisuuntaisen vuorovaikutuksen.

Oletetaan, että halutaan selvittää *frame_no* –attribuutin arvo. Tällöin selvitetään ensin intensionaalisen tason esityksestä se indeksi, johon *frame_no* liittyy. Vastaukseksi saadaan tässä tapauksessa indeksi t(1,3,1). Jos *frame_no*:oon olisi liittynyt useampia indeksejä, olisivat myös nämä indeksit tulleet vastaukseksi. Tämän jälkeen haetaan ekstensionaaliselta tasolta indeksiä t(1,3,1) vastaava arvo. Koska esimerkin yhteydessä on esiteltynä vain yksi ilmentymä, vastaukseksi tulee *4566545*, sillä se on rakenteessa indeksin osoittamassa paikassa.

Jos oltaisiin haluttu saada oliotyyppiä *frame* vastaavat ekstensionaalisen tason rakenteet, systeemi olisi etsinyt kaikki *frame*:en liittyvät indeksit intensionaalisen tason esityksestä. Niitä olisi ollut tässäkin tapauksessa jälleen vain yksi (indeksi t(1,3)). Tämän jälkeen ekstensionaaliselta tasolta olisi löytynyt indeksiä vastaava rakenne *tframe53617(4566545,steel,4)*. Jos halutaan käsitellä termin mukaista oliota, tarvitaan siihen tarkoitukseen suunnittelemaani predikaattia, joka muuttaa esityksen Prolog++ -mukaiseksi olioesitykseksi. Predikaatti muuntaa tuplen nimeä esittävän merkkijonon siten, että se poistaa t-kirjaimen nimestä ja irrottaa kirjainosuuden numero-osuudesta. Lopuksi esitys viimeistellään suluilla ja | -merkillä. Lopputulokseksi yllä olevasta *frame*-rakenteesta saadaan oliotunnus (frame|53617), jota käytetään viitattaessa muistissa olevaan olioon. Oliotunnuksen selvittämisen jälkeen oliota käsitellään normaalisti, koska kaikki sen metodit ja attribuutit ovat käytettävissä.

Siirryttäessä ekstensionaaliselta tasolta intensionaaliselle tasolle tarvitaan predikaattia, joka kykenee antamaan ekstensionaaliselta tasolta indeksit, jotka liittyvät tiettyihin arvoihin tai olioihin. Predikaatti tuottaa myös useamman indeksin tarvittaessa. Jos esimerkiksi halutaan saada arvoon *6* liittyvä intensionaalisen tason vastinkäsiteet selville, kolmipyörän ekstensionaalisen tason esityksestä, saadaan vastaukseksi indeksi t(1,6,4,1). Tämän jälkeen Prolog-ohjelma etsii indeksiä vastaavan nimen intensionaalista tasoa vastaavasta binäärirelaatioesityksestä. Vastaukseksi esimerkin yhteydessä saadaan attribuutti *diam*. On huomionarvoista, että arvo olisi voinut esiintyä useammassa kohtaa ekstensionaalisella tasolla, jolloin kaikkien kohtien indeksit olisivat tulleet vastaukseksi.

Prolog++ esitystapa edellyttää seuraavia suoritusvaiheita osakokonaisuussuhteita sisältäviä rakenteita toteutettaessa: Ensin mallinnetaan luokka normaalin olio-orientoituneisuuden mukaisesti Prolog++:lla. Tällöin mallintaja päättää, periytyykö luokka mahdollisesti jostain yläkäsitteestä, ja mitä attribuutteja ja metodeja luokalle määritellään. Tämän lisäksi luokalle määritellään sen mahdolliset osat. Osien tulee olla luokkia, jotka on määritelty

Prolog-ohjelmassa aikaisemmin kuin niiden kompositio. Tämän jälkeen johdetut attribuutit muodostetaan metodien avulla. Johdettujen attribuuttien vaatimuksena on se, että ne ovat yksipaikkaisia metodeja. Metodin ollessa yksipaikkainen sitä voidaan soveltaa Prolog++:ssa samalla tavalla kuin attribuuttejakin. Tällöin käyttäjän ei tarvitse tietää, onko kyseessä johdettu- vai normaali attribuutti. Edellä olleet tehtävät riippuvat kulloisestakin sovellutusalueesta, missä kyselykieltä tullaan käyttämään. Näiden vaiheiden lisäksi prototyyppitoteutuksessa on suoritettava manuaalisesti sekä tulostusmetodin määrittäminen luokkaan, että intensionaalisen esityksen laatiminen. Tulevaisuudessa tämä työ on tarkoitus saada käyttäjälle automaattiseksi tai graafisella käyttöliittymällä suoritettavaksi. Myös ensimmäisen pakollisen vaiheen tulostusmetodin saamista automaattiseksi tutkitaan jatkossa.

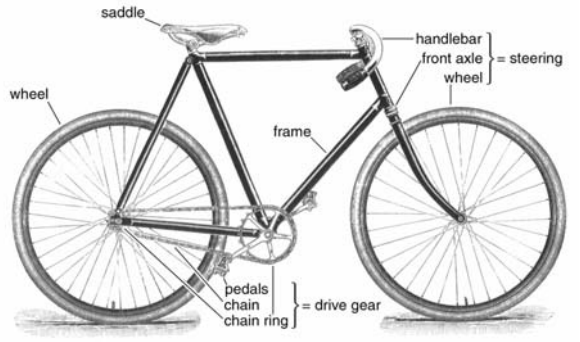
5.4.3. PSE-tietokannan kuvaus ja esimerkkitietokanta

Tietokannan tulee sisältää kaikkien ylimpien tasojen oliotyyppien (top-type) intensionaalinen kuvaus aiemmin esitetyllä tavalla. Oletetaan, että tietokanta koostuu kahdesta polkupyörätyypistä, aiemmin esitetystä *tricycle*:stä ja lisäksi Junkkarin [2001] työssä esitetystä *bicycle*:stä. Tällöin tietokantaan totetutetaan niihin liittyvät intensionaalisen tason esitykset. Intensionaalista tasoa ei siis ole tarkoitus esittää kaikista *tricycle*:n ja *bicycle*:n alikokonaisuuksista itsenäisinä kuvauksina. Jos sen sijaan pyörien alikokonaisuuksia luodaan itsenäisinä osina, tulee myös alikokonaisuuksien intensionaalinen esitys toteuttaa erikseen.

Seuraavaksi esittelen esimerkkitietokannan, joka koostuu kolmesta *tricycle*-oliosta ja kolmesta *bicycle*-oliosta. *Tricycle*-luokan osa-kokonaisuusrakenne on sama, kuin aikaisemmissa esimerkeissä. Tällöin *tricycle*:llä oli ainoastaan yksi ilmentymä. Tarkastelen myöhemmin esiteltäviä esimerkkikyselyjä tähän esimerkkitietokantaan perustuen. Esimerkkitietokannan toteutus muodostuu siis kahdesta intensionaalisen tason Prolog-esityksestä ja kuudesta ekstensionaalisen tason Prolog-esityksestä. Prolog-termeinä tietokantaa ei tässä yhteydessä enää esitetä, koska termit ovat toteutusrakenteita, joista kyselykielen käyttäjän ei tarvitse olle selvillä. Sen sijaan havainnollistan tietokantaa aikaisemmin käyttämälläni sisäkkäisten taulukoiden mukaisella visualisointitavalla. Visualisoinnista on jätetty tilankäytön vuoksi johdettu attribuutti *weigth* pois, vaikka se oletetaan esimerkkikyselyissä olevan mukana. *Weigth* on siis jokaisella PSE-esityksen kompositiolla johdettuna attribuuttina ja se on toteutettu metodin avulla. Perusoliotyypeillä *weigth* on normaali attribuutti. Ennen *bicycle*-osa-kokonaisuussuhteen taulukkoesitystä havainnollistetaan bicyclen rakenne kuvassa 5.

TRICYCLE																												
oid	Price	FRAME			SADDLE		STEERING												REAR									
		oid	Frame-No	Material	oid	Pad	oid	H.	FRONT AXLE		HANDLEBAR		PEDALS		WHEEL			oid	B.	REAR AXLE			WHEEL		
									oid	L.	oid	B.	oid	Diam.	oid	Diam.	R_type			oid	Diam.	L.	oid	Diam.	R_type
o12	100	o9	4566545	steel	o10	plastic	o11	17	o5	4	o6	13	o7	5	o8	8	united	o4	13	o1	0.5	12	o2	6	united
																							o3	6	united
o13	250	o14	6534264	steel	o15	leahter	o16	16	o17	3	o18	14	o19	7	o20	9	solid	o21	12	o22	0.6	13	o23	5	solid
																							o24	5	solid
o25	400	o26	5435534	aluminium	o27	rubber	o28	19	o29	4	o29	12	o30	6	o31	7	united	o32	14	o33	0.5	13	o34	7	united
																							o35	7	united

Kuva 4: *Tricycle*n intensionaalinen taso ja sen kolme ilmentymää.



Kuva 5: B*icycle*:n komponentit

BICYCLE																																		
oid	Price	FRAME				DRIVE GEAR											SADDLE			STEERING													WHEEL		
		oid	Frame-No	Mat.	W.	oid	C_type	CHAIN			CAIN RING			PEDALS			oid	Mat.	W.	oid	H.	FRONT AXLE			HANDLEBAR			WHEEL				oid	Diam	R_type
								oid	L.	W.	oid	Diam	W.	oid	Diam	W.						oid	L.	W.	oid	B.	W.	oid	Diam	R_type	W.			
o37	500	o27	8265	steel	10	o28	1-speed	o13	40	0.5	o14	10	0.5	o16	10	0.5	o29	leather	0.5	o30	39	o17	26	1	o18	20	1	o19	26	spoke	2	o31	26	spoke
											o15	4	0.2																					
o38	400	o32	43285	steel	8	o33	1-speed	o20	39	0.5	o21	9	0.5	o23	7	0.5	o34	plastic	0.5	o35	30	o24	20	1	o25	20	1	o26	20	spoke	2	o36	20	spoke
											o22	4	0.2																					
o39	400	o40	43277	steel	8	o41	1-speed	o42	39	0.5	o43	9	0.5	o45	8	0.5	o46	plastic	0.5	o47	30	o48	20	1	o49	20	1	o50	20	spoke	2	o51	20	spoke
											o44	4	0.2																					

Kuva 6: *Bicycle*:n intensionaalinen taso ja sen kolme ilmentymää.

Kun siis olio luodaan, se luo itsestään Prolog-esityksen, joka vastaa kuvissa 4 ja 6 olevalla visualisointitavalla esitettyä yhtä valkoisella visualisoitua riviä.

Tietokanta sisältää siis kaikki osa-kokonaisuussuhteiden intensionaaliset esitystavat. Tästä seikasta koituu ongelma silloin kun tietokannassa on paljon osa-kokonaisuussuhteita, joilla on samannimisiä komponentteja. Oletetaan tilanne, jossa tietokanta muodostuu kahdesta polkupyörätyypistä. ja halutaan tarkastella ainoastaan *tricycle*n rakennetta. Jos halutaan saada esimerkiksi *tricy-*

cle:n polkimien komposiitit kyselyn vastaukseksi, kyselyn tuloksena saataisiin molempien polkupyörätyyppien polkimien komposiitit. Ongelma esiintyy tilanteessa, jossa molempien polkupyörätyyppien polkimet on mallinnettu samalla nimellä. Tämän takia kieleen on toteutettu primitiivi, jolla käyttäjä voi määrätä, minkä kokonaisuuksien suhteista hän on kiinnostunut. Tällöin käyttäjä voi ottaa kyselyn tarkastelun kohteiksi esimerkiksi vain yhden osa-kokonaisuussuhteen. Oletuksena kyselyissä käytetään koko tietokannan sisältämiä osa-kokonaisuussuhteita. Ekstensionaalisella tasolla ongelmaa ei esiinny, sillä oliot ovat aina yksikäsitteisiä.

6. Kielen primitiivit ja esimerkkikyselyt

6.1. Kielen syntaksiin vaikuttavia tekijöitä

Kielen primitiivien tulee olla sellaisia, että niitä yhdistelemällä saadaan aikaan ilmaisuvoimaisia kyselyjä. Kieli on yritetty pelkistää mahdollisimman yksinkertaiseksi, jotta kieli olisi helppokäyttöinen. Kielen suunnittelua ohjasivat seuraavat periaatteet: helppokäyttöisyys, tietokoneen rajoitukset, riittävä ilmaisuvoima ja primitiivien mahdollisimman vähäinen määrä. Viimeinen vaatimus on siksi, että primitiivien ulkoa muistaminen ei koituisi vaikeaksi. Lisäksi primitiivien hyödyntäminen on vaikeaa jos käyttäjä ei hahmota, kuinka yhdessä kyselyssä voidaan ilmaista vaativakin kysely yhdistelemällä primitiivejä keskenään. Yllä olevat neljä vaatimusta ovat keskenään osittain ristiriitaisia, ja tämän vuoksi tehtävä ei ole helppo.

Kieli perustuu jaetun muuttujan ideaan ja korkealla abstraktiotasolla oleviin perusprimitiiveihin. Jaetun muuttujan idealla tarkoitetaan samaa, kuin logiikkaohjelmoinnin yhteydessä. Tämän lisäksi syntaksissa on käytettävissä logiikan käsitteet konjunktio, disjunktio sekä sulut. Näiden käsitteiden avulla tulee voida muodostaa kaikki kielen lailliset kyselyilmaisut.

Syntaksissa isolla kirjoitetut kirjaimet merkitsevät muuttujia ja pienellä kirjoitetut sanat kielen primitiivejä tai vakioita. Pilkku tarkoittaa konjunktiota, eli se merkitsee loogista käsitettä *ja*. Disjunktion tapauksessa käytetään puolipistettä ja se tarkoittaa loogista käsitettä *tai*. Sulkuja käytetään kyselyn sisäisen suoritusjärjestyksen vaihtamiseen.

Käyttäjän tulee kaikkiaan hallita siis edellä mainitsemani loogiset peruskäsitteet ja kielen perusprimitiivit. Kielessä olevat loogiset käsitteet ovat helppoja, eikä niiden omaksuminen edellytä logiikan opiskelemista. Jaetun muuttujan idea on myös helppo ymmärtää. Kielen primitiivien demonstroinnissa, käytän edellisessä kappaleessa esittelemääni esimerkkitietokantaa. Samaa esimerkkiä käytetään myös kyselyjen demonstroinnissa.

6.2. Kielen primitiivit

Kielen primitiivien tehtävänä on mahdollistaa itsenäisesti miellettävien kyselyn osien määrittely. Jos kysely on yksinkertainen, yhdenkin primitiivin käyttö voi riittää. Kysely muodostetaan primitiiveillä, joissa käytetään muuttujia ja vakioita. Tässä tutkielmassa primitiivin argumenteilla tarkoitan muuttujia tai vakioita, jotka ovat primitiivin käsittelyssä. Pilkulla ja puolipisteellä yhdistellään

primitiivit toisiinsa kyselyssä. Jos primitiivi käsittelee kahta argumenttia, argumentit sijoitetaan primitiivin kummallekin puolelle. Yhden argumentin primitiivin tapauksessa argumentti sijoitetaan primitiivin välittömään läheisyyteen jommalle kummalle puolen primitiiviä. Argumentti voi olla myös tietyissä primitiiveissä listarakenteinen. Lisäksi yksi primitiivi toimii relaatioalgebran valintaoperaation tapaan ja se esitellään myöhemmin tarkemmin.

Kielen primitiivit voidaan jakaa käyttötarkoituksensa perusteella kolmeen ryhmään: intensionaalisen tason- ja intensionaalis-ekstensionaalisen tason käsittelyprimitiiveihin sekä muihin primitiiveihin. Muut primitiivit eivät liity itse osa-kokonaisuussuhteen analysointiin. Niillä on silti tärkeä rooli kyselykielessä.

6.2.1. Intensionaaliset primitiivit

Kielessä on yhteensä kahdeksan intensionaalista primitiiviä. Alla luetellaan kyseiset primitiivit. *Arg1* ja *arg2* tarkoittavat primitiivien argumentteja.

1. arg1 **is_composite_type_of** arg2
2. arg1 **is_component_type_of** arg2
3. arg1 **is_top_type**
4. arg1 **is_basic_type**
5. arg1 **is_property_of** arg2
6. arg1 **is_path_to** arg2
7. **common_component** arg1
8. **common_components** arg1.

Kyselyjen oletetaan kohdistuvan sekä *tricycle*:n että *bicycle*:n osa-kokonaisuussuhteisiin, jos muuta ei mainita.

Ensimmäisen primitiivin pääkäyttötarkoitus on komposiittityypin etsiminen jollekin osalle. Primitiivin avulla saadaan selville sekä välilliset, että välittömät komposiittityypit. Argumentti voi olla muuttuja tai vakio. Vakion tulee olla jokin tietokannasta löytyvän oliotyypin nimi. Argumentti *arg1* tarkoittaa komposiittityyppiä, ja *arg2* sen komponenttityyppiä. Primitiivi arvottaa argumentit, jotka ovat muuttujia. Esimerkiksi tämän primitiivin ilmaisulla

X is_composite_type_of pedals.

X:n arvotuksena saadaan *pedals*-luokan komposiittiluokat. Toisin sanoen esimerkkitietokantaan soveltamalla primitiivi antaa *X*:lle seuraavat arvotukset:

X = steering, X = tricycle, X = drive-gear,X = bicycle

Primitiivillä voidaan saada selville myös jonkin osan komponentit soveltamalla primitiiviä alla olevalla tavalla. Siinä primitiivi arvottaa X:n kaikilla tietokannasta löytyvillä *steering*-luokan komponenteilla Ilmaisu

steering is_composite_type_of X.

tuottaa seuraavat arvotukset

X = front-axle, X = handlebar, X = pedals, X = wheel

Primitiiviä voidaan käyttää myös etsimään kaikki kompositio-/komponenttisuhteet, jotka tietokannassa löytyvät. Tämä saadaan aikaiseksi siten, että molemmat primitiivin argumentit ovat muuttujia. Esimerkiksi ilmaisu:

X is_composite_type_of Y.

antaa suuren joukon vastauksia (eli X:n ja Y:n arvotuksia), joista vain seuraavat esitetään

{X = steering, Y = front-axle}, {X = tricycle, Y = pedals} ja {X = bicycle, Y = saddle} jne…

Toinen primitiivi on ensimmäisen käänteisprimitiivi. Ensimmäisellä primitiivillä pystytään ilmaisemaan samat asiat kuin toisella. Primitiivi on kuitenkin toteutettu, jotta käyttäjä voi käyttää kieltä intuitiivisemmin.

Kolmas primitiivi antaa tietokannasta osa-kokonaisuussuhteen ylimmän tason tyypit. Esimerkkitietokannan tapauksessa ilmaisu

X is_top_type

antaa vastaukseksi

X = tricycle, X = bicycle.

Neljäs primitiivi antaa puolestaan tietokannassa olevien osa-kokonaisuussuhteiden perustyypit, ja sitä käytetään argumentin osalta samaan tapaan kuin *is_top_type* -primitiiviä.

Viidennellä primitiivillä saadaan selville oliotyypin ominaisuudet. Argumentilla *arg1* ilmaistaan ominaisuus, joka liittyy argumentilla *arg2*:lla ilmaistuun oliotyyppiin.Esimerkkitietokannasta kysely

X is_property_of wheel

antaa tulokseksi

X = Diam, X = Rtype ja X = W.

Kuudes primitiivi antaa argumenttiin *arg2* johtavan polkuesityksen argumentissa *arg1*. Primitiivi antaa kaikki polkuesitykset, joita tietokannasta oliotyypille löytyy. Kysely

X is_path_to wheel

antaa esimerkkitietokannasta vastaukset

X = t(wheel,steering,tricycle), X = t(wheel,rear,tricycle), X = t(wheel,steering, bicycle) ja X = t(wheel,bicycle).

Seitsemännessä primitiivissä muuttuja arvottuu sellaisilla oliotyyppeillä, jotka esiintyvät kaikissa tietokannassa olevissa osa-kokonaisuussuhteissa komponentteina. Kahdeksas primitiivi on samanlainen kuin seitsemäs, mutta vastaus saadaan listaesityksenä. Primitiiviä seitsemän kannattaa käyttää tilanteissa, joissa primitiivin tulosta käsitellään kyselyssä jonkin toisen primitiivin argumenttina. Kahdeksas primitiivi on kätevä tilanteissa, joissa primitiivin tulosta ei käsitellä edelleen, sillä listaesitys on havainnollisempi.

6.2.2. Intensionaalis-ekstensionaaliset primitiivit

Intensionaalisen ja ekstensionaalisen tason yhdistäviä primitiivejä on kielessä kuusi. Tasojen yhdistäminen ilmaistaan siten, että primitiivin toinen argumentti kuuluu ekstensionaaliselle tasolle ja toinen intensionaaliselle tasolle. Kieli sisältää seuraavat intensionaalis-ekstensionaaliset primitiivit:

1. arg1 **is_instance_of** arg2
2. arg1 **:** arg2
3. arg1 **is_composite_object_of** arg2
4. arg1 **is_component_object_of** arg2
5. arg1 **is_basic_object**
6. arg1 **is_top_object**

Ensimmäinen primitiivi ilmaisee luokan ja sitä vastaavan ilmentymän. Argumenttina *arg1* on olio (ekstensionaalinen taso), joka kuuluu oikealla puolella olevaan luokkaan (intensionaalinen taso). Luokka ilmaistaan argumentilla *arg2*. Alla on kolme esimerkkiä primitiivin käytöstä.

1. X is_instance_of saddle
2. (tricycle | 543664) is_instance_of X
4. X is_instance_of Y.

Ensimmäisessä esimerkissä saadaan esimerkkitietokannasta oliot, joilla on olioidentiteetit *o10*,*o15*,*o27*,*o29*,*o34* ja *o46*. Toisessa esimerkissä primitiiville annetaan Prolog++ -esitystavan mukainen oliorakenne ja vastaukseksi saadaan *tricycle*. Tämänkaltaista kyselyä ei ole järkevää tehdä, mutta jos on kyse muuttujasta joka on alustettu olioksi, on tällaisesta primitiivin käyttötavasta useissakin tilanteissa hyötyä. Kolmannessa esimerkissä saadaan vastaukseksi kaikki tietokannan oliot ja niitä vastaavat oliotyypit.

Toinen primitiivi on tarkoitettu olion ominaisuuksien arvojen selvittämistä tai tietyillä ominaisuuden arvoilla olevien olioiden etsimistä varten. *Arg1* on olio ja arg2 on muotoa *property(arg3)* oleva termi, jossa property on sen oliotyypin ominaisuus, johon olio *arg1* kuluu. *Arg3* voi olla joko muuttuja tai attribuutin arvo. Alla olevat esimerkit selventävät primitiivin käyttöä:

1. X:material(Y)
2. X:material(aluminium)

Ensimmäisessä esimerkissä primitiivin käytölle on ehtona, että muuttuja on alustettu olioksi primitiivin käyttöhetkellä. Ensimmäinen esimerkki arvottaa Y:n X-olion *material*-attribuutin arvolla. Jos X on arvotettu esimerkkitietokantamme *tricycle*:n komponenttina olevalla *frame*-oliolla, jonka id on o14, saadaan vastaukseksi *Y = steel* . Toisessa esimerkissä primitiiviä käytetään antamaan rajoite. Tarkasteltavan olion täytyy siis täyttää ehto, että sen *material*-ominaisuuden arvona on *aluminium*. Tällöin ehdon täyttää esimerkkitietokan-

nan olio, jolla on oliotunnus *o26*. Jos oliolta kysytään ominaisuutta jota sillä ei ole, sulkujen sisällä oleva muuttuja saa arvon *null*. Primitiiviä voi käyttää myös siten, että ominaisuuteen viitataan muuttujalla, mutta esittelen tätä ominaisuutta myöhemmin esimerkkikyselyjen yhteydessä.

Kolmas primitiivi ilmaisee oliotyypin komposiittityyppien oliot. *Arg1* on olio, joka kuuluu *arg2*:lla ilmaistuun oliotyypin komposiittityyppiin. Täten kysely

X is_composite_object_of pedals

antaa vastaukseksi esimerkkitietokannasta oliot seuraavilla oliotunnisteilla:

o11,o16,o28,o12,o13,o25,o28,o33,o41,o37,o38, o39.

Kolmannella primitiivillä ei voi etsiä komponenttiobjekteja. Ne siis eivät ole toistensa käänteisprimitiivejä, vaikka intensionaalisten primitiivien kohdalla *is_composite_type_of* ja *is_component_type_of* –primitiivit olivatkin. Tämä johtuu siitä, että primitiivin argumentit ovat eri tason argumentteja. Primitiiviä voi käyttää myös siten, että *arg1* on alustettu olioksi. Tällöin *arg2* alustetaan *arg1*:stä vastaavan oliotyypin komponenttityypillä. Myös molemmat argumentit voivat olla muuttujia. Tällöin primitiivi hakee tietokannasta kaikki oliot ja niitä vastaavien oliotyyppien komponenttityypit.

Neljättä primitiiviä käytetään samaan tapaan kuin 3. primitiiviä, mutta se etsii annetun oliotyypin komponenttioliot.

Viides ja kuudes primitiivi etsivät tietokannan osa-kokonaisuussuhteiden ylimmän tason oliot ja perusoliot vastaavassa järjestyksessä. Niitä käytetään saman tapaan kuin intensionaalisia vastinprimitiivejäänkin, mutta ne palauttavat oliot tyyppien sijaan.

6.2.3. Kyselyn tuloksen esittämisprimitiivit

1. arg1 **Where** arg2
2. **Min()**
3. **Max()**
4. **Count()**
5. **Avg()**

Tämä ryhmä koostuu primitiiveistä, joilla voidaan vaikuttaa kyselyn tuloksen muotoon ja tuottaa erilaisia aggregointitietoja kyselyjen vastauksiin. Aggre-

gointitiedot ilmaistaan aggregointioperaatioilla, joita ovat yllä olevassa luettelossa operaatiot 2-5. Nämä operaatiot on lainattu deduktiivisille oliotietokannoille suunnitellusta Prolog++ -pohjaisesta kyselykielen prototyypistä [Christensen, 1998]. Tämä siksi, että ne sopivat lähes sellaisinaan tutkielman kyselykielen tarpeisiin.

Where-primitiivi (alkuperäiseltä nimeltään *provided*) on kyselyn muodostamisen kannalta keskeinen. Primitiivin argumentilla *arg1* käyttäjä ilmaisee tulosrelaation muodon, johon hän poimii haluamansa muuttujat oikealla puolella olevasta kyselyilmauksesta. *Where*-primitiivi muistuttaa relaatioalgebran valintaoperaatiota, jolla halutut relaation attribuutit valitaan tulosjoukkoon. *Where*-primitiivin soveltamisella valitaan kuitenkin relaation attribuuttien sijasta kyselyssä mukana olevia muuttujia. Tällöin voidaan jättää vastauksesta kyselyssä ilmenevät, mutta loppukäyttäjän näkökulmasta merkityksettömät muuttujat pois. *Where*-primitiivillä on kuitenkin monipuolisemmat käyttömahdollisuudet, mitä relaatioalgebran valinta-operaattorilla. *Where* –primitiivin vasemmalle puolelle tuleva tulos ilmaistaan tuple-rakenteina. Käyttäjä voi itse valita tuloksessa käytettävän tuplen nimen. Jos tuplen muuttujat on arvotettu olioiksi olioita, niihin voi soveltaa *:* -primitiiviä, jolla olion tietyn attribuutin arvo saadaan tulokseen. Primitiivin käyttötapa tulosjoukon yhteydessä eroaa kuitenkin edellä esitetystä. Relaatiossa ei nimittäin käytetä sulkuja ominaisuuden nimen jälkeen. Pelkkä ominaisuuden nimi siis riittää tulosjoukon yhteydessä. Jos tuloksessa ilmaistaan *:* -primitiivi oliolle, jolla ei ole kyseistä ominaisuutta, niin vastaukseksi tähän kohtaan tuple-rakenteessa saadaan arvo *null*.

Where-primitiivin *arg1*:ssä ilmaistut tuplen alkioina olevat muuttujat arvottuvat joillakin arvoilla. Vastaus ilmaistaan relaationa ja relaation tuplet tulostetaan omille riveilleen allekkain. Primitiivi poistaa kaikki samat tuplet, relaatiomallin tapaan. Esimerkiksi kysely

tulos(X) where X is_component_type_of steering

tulostaa vastaukseksi esimerkkitietokannasta seuraavat yksipaikkaiset tulostuplet. Toisin sanoen muuttuja X on arvottunut where-primitiivin oikealla puolella olevan kyselyilmauksen mukaisesti.

tulos(front_axle)
tulos(handlebars)
tulos(pedals)
tulos(wheel)

Seuraavassa kyselyssä käytetään : - primitiiviä tulosrelaation muodon määrittelyssä. Kysely antaa tulokseksi kaikki ylimmän tason olioiden tyypit ja näiden tyyppien ilmentymiä vastaavat *price*-attribuutin arvot:

tulos(X,Y:price) where X is_top_type,Y is_instance_of X

Kysely antaa vastaukseksi esimerkkitietokantaan liittyen seuraavan relaation:

tulos(tricycle,100)
tulos(tricycle,250)
tulos(tricycle,400)
tulos(bicycle,500)
tulos(bicycle,400)

Sama tulos oltaisiin saatu myös kyselyllä

tulos(X,Y) where X is_top_type,Z is_instance_of X,Z:price(Y).

Aikaisempi kysely oli kuitenkin yksinkertaisempi: vastaukseen on helppo poimia jokin muuttuja, joka tarkoittaa oliota ja viitata sitten sen ominaisuuden arvoon. Tulosrelaation määrittelyssä voidaan viitata sekä tavalliseen, että johdettuun attribuuttiin. Jos sen sijaan olioille asetetaan rajoituksia attribuuttien arvojen perusteella, joudutaan käyttämään ominaisuuden arvon ilmaisemista jälkimmäisellä tavalla.

Aggregointifunktiolla suoritetaan erilaisia tietokannassa oleviin tietoihin kohdistuvia laskutoimituksia. Aggregointioperaatioita ovat lukumäärä *count*, keskiarvo *avg* sekä maksimi- *max* ja minimiarvon *min()*. Tulosrelaation kuvauksessa ilmaistaan aggregointioperaatioiden argumenttina se kohde, josta aggregointitietoja halutaan laskea. Kysely

tulos(count(X)) where X is_top_object.

tulostaa vastaukseksi

tulos(6)

Kyselyn tuloksena saatiin esimerkkitietokannassa olevien osa-kokonaisuussuhteiden ylimpien tasojen olioiden kokonaislukumäärä. Vaikka

täsmälleen samat rivit poistetaankin tulosrelaatiosta, aggregoinnissa ne otetaan huomioon, jotta saadaan oikea tulos.

Kyselyssä ei ole rajoitetta aggregointifunktioiden ilmaisemisen määrää. Seuraavassa kyselyssä halutaan osa-kokonaisuussuhteiden ylimmän tason olioiden lukumäärä, minimipaino, maksimipaino ja keskimääräinen paino.

tulos(count(X),min(Y:weigth),max(Y:weigth),avg(Y:weigth)) where X is_top_type,Y is_instance_of X.

Tuloksena saadaan

tulos(6,10.1,17,14.3333333333333).

Pilkut erottavat tuloksen argumentit toisistaan ja desimaaleja merkitään pisteillä.

6.2.4. Muut primitiivit ja komennot

Tähän ryhmään kuuluvat primitiivit, jotka eivät kuulu mihinkään edellä esitettyyn ryhmään. Ensimmäinen primitiivi on varsinainen kyselyprimitiivi, muut tietokantasovelluksen käsittelyyn liittyviä primitiivejä.

1. **Apply_to** arg1
2. start
3. add
4. quit

Apply_to-primitiivi rajoittaa kyselyssä tarkasteltavia osa-kokonaisuussuhteita. Kyselyissä oletusarvona on se, että kysely kohdistuu kaikkiin osa-kokonaisuussuhteisiin. *Arg1* sisältää listaesityksenä niiden ylimpien tyyppien nimet, joihin kyselyssä ilmaistujen primitiivien halutaan kohdistuvan. Jos käyttäjä haluaa esimerkkitietokannan tapauksessa sekä *tricycle*:n-, että *bicycle*:n osa-kokonaisuudet mukaan kyselyihinsä, hän käyttää seuraavaa ilmaisua:

apply_to [tricycle,bicycle].

Apply_to –primitiivin vaikutus on voimassa vain kyselyn ajan ja seuraavassa kyselyssä on taas kaikki osa-kokonaisuussuhteet mukana.

Start-komento käynnistää kyselytilan, jonka jälkeen kyselyitä voi suorittaa. Quit-komento siirtyy puolestaan pois kyselytilasta. Add-komennolla voidaan luoda uusia olioita tiedostoon. Tämä helpottaa olioiden luomista, sillä kompleksisen rakenteen omaavia olioita luotaessa täytyy asettaa monta arvoa. Olioiden luonnissa tarvittavat arvot voidaan kirjoittaa vain kerran tiedostoon. Kun add-komento suoritetaan, ohjelma pyytää käyttäjää kirjoittamaan tiedoston nimen, jossa olion luontikomennot sijaitsevat.

6.3. Esimerkkikyselyt

Olen jaotellut esimerkkikyselyt samalla tavalla kuin kappaleessa neljä. Annan toisin sanoen esimerkkikyselyjä intensionaalisista-, ekstensionaalisista- sekä intensionaalis-ekstensionaalisista kyselyistä. Jokaisen kyselytyypin kohdalla esitän myös kyselytyyppiä vastaavia yhdistettyjä kyselyjä.

6.3.1. Intensionaaliset esimerkkikyselyt

Ensimmäisellä tämän kategorian esimerkkikyselyllä haetaan *tricycle*:n peruskomponenttityypit seuraavasti:

```
result(X) where apply_to[tricycle], X is_basic_type.
```

Kyselyn alussa määritellään, että ollaan kiinnostuneita ainoastaan *tricycle*:n osakokonaisuussuhteesta. Vastaus saadaan kyselyn muuttujan X arvotuksista. Vastaukseksi saadaan tulokset

```
result(frame)
result(saddle)
result(front_axle)
result(handlebar)
result(pedals)
result(rear_axle)
result(wheel).
```

Jos kyselyn haluttaisiin kohdistuvan koko esimerkkitietokantaan, olisi kyselystä pitänyt jättää *apply_to* –primitiivi pois. Jos kyselyn kohdassa *result(X)* olisi sovellettu *count()* –aggregointiprimitiiviä, oltaisiin saatu selville kuinka monta peruskomponenttityyppiä *tricycle*:llä on.

Seuraavalla kyselyllä haetaan kaikkien niiden oliotyyppien polut, joilla on ominaisuutena *diam*.

res(X) where diam is_property_of Y,X is_path_to Y.

Kyselyssä määritellään ensin, että muuttujan Y tulee olla oliotyyppi, jolla on ominaisuus *diam*. Tämän jälkeen ilmaistaan, että X on Y:hyn johtava polku. Vastauksena saadaan siis tuple-muotoiset polkuesitykset poluista, jotka johtavat oliotyyppeihini *rear_axle*, *chainring*, *pedals* ja *wheel*. Jos tietyllä tyypillä on useampi polkuvaihtoehto, kaikki vaihtoehdot saadaan vastaukseksi. Ylläolevaan kyselyyn tuotetaan seuraava vastaus esimerkkitietokannan yhteydessä.

res(t(tricycle,rear,rear_axle))
res(t(bicycle,drivegear,chainring))
res(t(tricycle,steering,pedals))
res(t(bicycle,drivegear,pedals))
res(t(tricycle,steering,wheel))
res(t(tricycle,rear,wheel))
res(t(bicycle,wheel))
res(t(bicycle,steering,wheel))

Tricycle:n ohjauksen komponenttien nimet ja niiden ominaisuuksien nimet saadaan selville seuraavalla kyselyllä:

res(X,Y) where apply_to [tricycle],X is_component_type_of steering, Y is_property_of X.

Ensin ilmaistaan, että tarkastelu rajoitetaan vain *tricycle*:n osa-kokonaisuussuhteeseen. Tämän jälkeen ilmaistaan, että X-muuttuja alustetaan ohjauksen komponentilla ja että Y-muuttuja on X-muuttujan sisältämän komponentin ominaisuus. Tulosrelaatio koostuu X- ja Y-muuttujien seuraavista arvoista.

res(front_axle,l)
res(front_axle,weigth)
res(handlebar,b)
res(handlebar,weigth)
res(pedals,diam)
res(pedals,weigth)
res(wheel,diam)
res(wheel,r_type)

res(wheel,weigth)

Seuraavaksi esitän yhdistetyn kyselyn, jossa intensionaalisen vastauksen antavan kyselyn sisällä ilmaistaan ekstensionaalinen alikysely. Kyselyssä halutaan saada vastaukseksi intensionaalista tietoa käyttämällä ekstensionaalista kriteeriä. Kyselyssä etsitään kaikki ne *tricycle*:n ja *bicycle*:n yhteiset komponenttityypit, joiden ekstensionaaliselta tasolta löytyy ainakin yksi olio, jonka material-attribuutin arvo on steel.

res(X) where common_component X, Y is_instance_of X, Y:material(steel).

Kyselyssä siis ilmaistaan, että X on yhteinen komponentti kaikille tietokannassa oleville osa-kokonaisuussuhteille. Esimerkkitietokannassa ei ole muita osa-kokonaisuussuhteita kuin *tricycle* ja *bicycle*. Tämän takia apply_to –primitiivin käyttö ei ole välttämätöntä, koska oletuksena kyselyyn otetaan kaikki osa-kokonaisuussuhteet. Muuttuja Y ilmaisee kyselyssä yhteisen komponentin mitä tahansa ilmentymää. Lopuksi kyselyssä rajoitetaan Y –muuttujan sisältämiä olioita siten, että olion *material*-attribuutin arvona täytyy olla *steel*. Kyselyn tulokseen halutaan saada vain X-muuttujan arvot. Tuloksia saadaan esimerkkitietokannan yhteydessä vain yksi:

res(frame)

Seuraava intensionaalinen kysely on myös yhdistetty kysely. Siinä etsitään kaikki ne *tricycle*:n komponenttityypit, joilla on ainakin yksi ilmentymä, jonka *diam*-attribuutin arvona on joko 4 tai 5.

res(Y) where X is_component_object_of tricycle, (X:diam(4);X:diam(5)), X is_instance_of Y.

Kyselyssä ilmaistaan, että X-muuttujan tulee olla kolmipyörätyypin komponenttiolio, jonka *diam*-attribuutin arvon täytyy olla joko 4 tai 5. Lisäksi ilmaistaan, että muuttuja Y on X-komponenttia vastaava oliotyyppi. Vastaukseen poimitaan ainoastaan Y-muuttujan arvotuksia. Vastaukseksi esimerkkitietokannassa saadaan kaksi oliotyyppiä:

res(pedals)
res(wheel)

6.3.2. Ekstensionaaliset kyselyt

Ekstensionaalisilla kyselyillä tarkoitetaan kaikkia sellaisia kyselyjä, jotka antavat ainoastaan ekstensionaalista tietoa vastauksenaan. Tämän vuoksi olioita vastauksena antavat kyselyt kuuluvat myös tähän kyselyryhmään. Käyttäjä ei ole tavallisesti kiinnostunut olioidentiteeteistä. Olioidentiteetit ovat ainoastaan yksilöinnin välineenä olio-orientoituneessa toteutuksessa. Tämän vuoksi olioita vastauksenaan antavia kyselyjä ei käsitellä, vaikka sellaisten kyselyiden ilmaiseminen kielellä onkin mahdollista.

Seuraava kysely antaa *tricycle*:n ja *bicycle*:n komponentteina olevien polkimien *diam*-attribuutin arvot:

res(X:diam) where (X is_component_object_of bicycle; X is_component_object_of tricycle), X is_instance_of pedals.

Kyselyssä ilmaistaan, että muuttujan X tulee olla joko *bicycle*:n tai *tricycle*:n alikomponenttiolio ja lisäksi olion tulee kuulua *pedals*-luokkaan. Vaikka *bicycle*:ssä *pedals*-komponentti sijaitsee *drive gear*-komponentin komponenttina ja *tricycle*:ssä puolestaan *steering*-komponentin komponenttina, vastaus sisältää molemmat tapaukset. Tämäntapainen kysely ei olisi ollut mahdollista iteraatioon perustuvan kielen avulla, koska navigointipolku *pedals*-komponenttiin on erilainen *tricycle*:llä ja *bicycle*:llä. Vastaukseksi kyselylle esimerkkitietokannasta saadaan seuraava yksipaikkainen relaatio:

res(5)
res(6)
res(7)
res(8)
res(10)

Seuraavassa kyselyssä ollaan kiinnostuneita vain tiedosta, mitä *material*-attribuutin eri arvoja on *tricycle*-luokan ilmentymien komponenteilla olemassa. Kysely edustaa tilannetta, jossa ei ole merkitystä tiedolla, minkä tyyppisessä komponentissa kukin arvo sijaitsee. Kysely voidaan muodostaa seuraavalla ilmauksella:

res(X:material) where X is_component_object_of tricycle.

Käyttäjän ei tarvitse välittää siitä, onko komponentilla *material*-nimistä attribuuttia.

res(steel)
res(aluminium)

Seuraavassa kyselyssä ilmaistaan *tricycle*:n ilmentymien määrän ja niiden painon keskiarvon antava kysely:

res(avg(X:weigth),count(X)) where X is_instance_of tricycle.

Kyselyn lopputuloksessa halutaan saada johdetun attribuutin keskiarvo, eli kyselyn aikana suoritetaan ensin komponenttien painojen yhteenlasku kolmipyörän *weigth*-metodin avulla ja tämän jälkeen lasketaan näistä tuloksista vielä keskiarvo. Samalla voidaan kysyä, kuinka monta *tricycle*:ä tietokannassa on. Vastaus ilmaisee, että kannan kolmipyörien keskiarvo on 7.1 ja että keskiarvo laskettiin kolmesta *tricycle*-oliosta. Toisin sanoen kysely antaa seuraavan vastauksen:

res(7.1,3)

Seuraava esimerkki esittelee yhdistetyn kyselyn, joka antaa ekstensionaalisen vastauksen. Siinä halutaan saada selville sellaisen *bicycle:n frame*- komponentin materiaali, jonka *frame*:n sarjanumero on 5435534. Tässä kyselyssä käyttäjän ei tarvitse tietää, minkä niminen attribuutti *frame*:n sarjanumeroon liitetään. Kyselyssä

res(Y:material) where Y is_component_object_of tricycle, P is_property_of frame,Y:P(5435534).

käyttäjä ilmaisee, että Y tarkoittaa oliota, joka on *tricycle*:n alikomponenttina. Lisäksi hän ilmaisee, että P tarkoittaa *frame*:n ominaisuutta ja rajoittaa Y olion koskemaan vain oliota, jonka ominaisuuden arvona on 5435534. Kieli mahdollistaa siis erittäin käyttäjäystävällisen tavan viitata ominaisuuksiin. Riittää, että käyttäjä tietää ominaisuuden arvon. Hän voi tällöin kyselyssä viitata oliotyypin ominaisuuteen muuttujalla. Vastaukseksi yllä olevaan kyselyyn saadaan

res(aluminium).

Jos käyttäjä ei ole varma, viittaako arvo *5435534* oikean nimiseen attribuuttiin, hän voi tarkistaa sen muuttamalla yllä olevan kyselyn vastauksen muodostamisilmaisua seuraavalla tavalla: *res(Y:material,P).* Tällöin vastauksesta olisi selvinnyt, mihin attribuuttiin arvo liittyy. Tällainen kysely olisi ollut intensionaalis-ekstensionaalinen kysely, koska vastauksessa olisi ollut molempien tasojen tietoa.
Viimeinen ekstensionaalinen esimerkkikysely on myös yhdistetty kysely. Siinä kyselyllä analysoidaan, kuinka paljon kaikki *bicycle*:jen teräksiset komponentit painavat yhteensä.

res(sum(Comp:weigth)) where material is_property_of C, Comp is_instance_of C,Comp is_component_object_of bicycle, Comp:material(steel).

Kyselyssä ilmaistaan, että *Comp*-muuttujan tulee olla olio, jolla on *material* –niminen attribuutti ja jonka arvona on *steel*. Lisäksi komponentin tulee olla *bicycle*:n komponenttina. Kyselyn vastaus on

res(9).

6.3.3. Intensionaalis-ekstensionaaliset kyselyt

Tämä kyselytyyppi sisältää kyselykielen monipuolisimmat kyselyt. Vastauksena saadaan kummankin abstraktiotason tietoja, ja lisäksi tämäkin kyselytyyppi sisältää monia mielekkäitä yhdistettyjä kyselyjä. Kyselytyypin vastaukset ovat monissa tilanteissa havainnollisempia kuin edellä esitetyt kyselytyypit. Tämä johtuu siitä, että arvoja voidaan havainnollistaa arvoa vastaavalla intensionaalisella tasolla. Primitiivien esittelyn yhteydessä muodostinkin jo yksinkertaisen intensionaalis-ekstensionaalisen kyselyn. Siinä hain osa-kokonaisuussuhteen ylimmän tason oliotyypit ja tyyppejä vastaavien olioiden *price*-attribuutin arvo. Kysely on siitä kätevä, että käyttäjän ei tarvitse tehdä kyselyä jokaiselle ylimmän tason oliotyypille erikseen. Lisäksi vastauksesta on helppo vertailla eri tyyppien hintoja. Eri tyyppisten polkupyörien keskihinta olisi ollut helposti ilmaistavissa soveltamalla kyselyn vastaukseen *avg* –aggregointifunktiota.

Seuraavalla kyselyllä haetaan kaikkien *tricycle*:n komponenttien nimet ja näitä komponentteja vastaava keskimääräinen paino:

res(I,avg(X:weigth)) where X is_component_object_of tricycle, X is_instance_of I.

Kyselyssä ilmaistaan, että muuttuja X on *tricycle*:n komponenttiolio ja muuttuja I on X-muuttujaa vastaava tyyppi. Tulosrelaatio muotoillaan siten, että ensimmäisenä relaation attribuuttina on oliotyyppi. Sitä seuraa tyypin ilmentymien keskipaino. Tämänkaltaisilla kyselyillä voidaan luoda kokonaisvaltainen katsaus *tricycle*:jen komponenttien painon jakautumiseen eri osien kesken. Vastaukseksi kyselyyn saadaan relaatio

res(frame,3)
res (saddle,0.4)
res (steering,2.12)
res (rear,2.38)
res(front_axle,0.92)
res (handlebars,0.9)
res (pedals,0.34)
res (rear_axle,1.14)
res (wheel,1.29333333333333).

Laajennan seuraavaksi aikaisemmin esitettyä ekstensionaalista kyselyä, jossa oltiin kiinnostuneita siitä, mitä eri materiaaleja *tricycle*:n komponenteilla on olemassa. Tällä kertaa kysely kohdistetaan käsittelemään *bicycle*:ä. Materiaalivaihtoehtojen lisäksi seuraavalla kyselyllä saadaan selville, mihin komponenttiin materiaali liittyy. Kyselyn

res(I, X:material) where apply_to [bicycle],X is_component_object_of bicycle,X is_instance_of I,material is_property_of I.

vastaukseksi saadaan relaatio

res(frame,steel)
res(saddle,leather)
res(saddle,plastic).

Laajennan edellistä kyselyä entisestään. Nyt ilmaistaan kysely, jossa jaotellaan komponentit materiaalin perusteella ja lasketaan jaottelun mukaisten komponenttien yhteispaino esimerkkitietokannasta.

res(C,Material,sum(Comp:weigth)) where apply_to [bicycle], material is_property_of C, Comp is_instance_of C,Comp is_component_object_of bicycle, Comp:material(Material).

Kyselyssä where-primitiivin oikealla puolella ilmaistaan, että kyselyä sovelletaan ainoastaan *bicycle*:n osa-kokonaisuussuhteille *apply_to*-primitiivin avulla. *C*-muuttujalla tarkoitetaan komponenttityyppiä, jolla on attribuutti nimeltään *material*. Lisäksi ilmaistaan, että *Comp*-muuttuja tarkoittaa *C*-muuttujan mukaista ilmentymää ja että ilmentymän tulee olla *bicycle*:n alikomponenttioliona. Lopuksi *Comp*-muuttujan arvona olevan olion *material*-attribuutin arvoon viitataan *Material*-muuttujalla. Vastausrelaatioon halutaan mukaan komponenttityyppi, tyyppiä vastaava materiaali ja komponentti-materiaali -paria vastaavien komponenttien yhteenlaskettu paino. Komponenttityypin tieto löytyy siis *C*-muuttujasta ja tyyppiin liittyvä materiaali *Material*-muuttujasta. Soveltamalla *sum* –aggregointifunktiota, yksittäisen komponenttiolion painoon, saadaan kaikkien ehdot täyttävien komponenttien painojen summa selville.
Kyselyn vastauksena saadaan relaatio

res(frame,steel,10.2)
res(saddle,leather,1.1)
res(saddle,plastic,2.2).

Analyyttistä voimaa sisältyy myös kyselyyn, jossa saadaan vastaukseksi polkupyörän tietyn osan kaikki komponentit, niiden ominaisuudet ja ominaisuuksien arvot. Tällöin saadaan kerralla kattava näkemys tietyn komponentin intensionaalisesta ja ekstensionaalisesta tasosta. Seuraavassa esimerkkikyselyssä halutaan saada selville *tricycle*:n *steering*:in kaikki alikomponentit ja niitä vastaavat ominaisuudet. Lisäksi jokaisen ominaisuuden kohdalla halutaan vielä ominaisuuden arvo. Kysely

res(T,P,Z) where apply_to [tricycle], X is_component_object_of steering,X is_instance_of T,P is_property_of T,X:P(Z).

tuottaa laajan relaation, jonka tupleista esittelen vain muutaman:

result(front_axle,l,4)
result(front_axle,weigth,14)
result(handlebars,weigth,1.1)

result(handlebars,b,13)
result(pedals,weigth,3)
result(pedals,diam,5)
result(wheel,weigth,1.4)
result(wheel,diam,8)
result(wheel,r_type,united)

7. Yhteenveto

Tutkielmassa tarkastelin osa-kokonaisuussuhteen käsittelyyn soveltuvan kyselykielen suunnittelua ja sen toteutusta. Toteutin kielen Timo Niemen ja Marko Junkkarin suunnittelun pohjalta. Kielen ilmaisuissa ei tarvitse hallita olio- tai deduktiivisten tietokantojen yhteydessä käytettyjä, tavalliselle loppukäyttäjälle hankalia käsitteitä. Näitä ovat esimerkiksi rekursio, mallinsovitus ja iterointi. Tällöin loppukäyttäjän ei tarvitse omata ohjelmointitaitoja. Kielen avulla hänen ei myöskään tarvitse navigoida osa-kokonaisuussuhteissa. Kielen käyttäjän ei myöskään tarvitse hallita nest- ja unnest-operaatioiden tapaisia osa-kokonaisuushierarkian uudelleenstrukturointeja. Sen sijaan kielen primitiivit tukevat haluttujen komponenttien ja komposiittien välillä vallitsevien suhteiden monipuolista ilmaisemista. Nämä primitiivit hoitavat niin välillisten kuin välittömienkin komponenttien ja komposiittien käsittelyt. Tämän johdosta käyttäjän muistamistaakka kaaviotasosta vähenee ja osien ja kokonaisuuksien välillä kulkeminen helpottuu.

Johdetut attribuutit on toteutettu siten, että käyttäjän ei tarvitse tietää, kuinka arvo muodostuu. Kielessä johdettuja ja tavallisia attribuutteja käytetään samalla tavalla. Kielen deklaratiivisuuden aste on korkea. Kyselyt suoritetaan kuvailemalla primitiivien avulla yhteyksiä muuttujien välillä. Käyttäjän tulee ymmärtää vain jaetun muuttujan, disjunktion, konjunktion, sulkujen ja primitiivien merkitys. Tämän lisäksi käyttäjän tulee omata käsitys intensionaalisesta ja ekstensionaalisesta tasosta.

Kyselyillä voidaan tuottaa intensionaalista, ekstensionaalista tai intensionaalis-ekstensionaalista tietoa. Pelkästään ekstensionaaliset vastaukset riittävät tietyissä tilanteissa. Intensionaalisilla vastauksilla voidaan puolestaan saada tietoa käyttäjälle vieraista rakenteista. Kysely voidaan tämän johdosta suorittaa, vaikka osa-kokonaisuushierarkia luokkien nimineen ja attribuutteineen ei olisikaan täysin selvillä. Lisäksi intensionaaliset kyselyt voivat havainnollistaa ekstensionaalisen tason tietoja [Motro, 1994]. Jos esimerkiksi halutaan kyselyllä selvittää, mitkä komponentit voivat olla materiaaliltaan muovia, saadaan vastaukseksi intensionaalista tietoa. Tieto on tällöin kuitenkin ekstensionaalisen kriteerin mukaan tuotettu. Tämä on monesti paljon havainnollisempi ja tarkoitustaan paremmin palveleva vastaus kuin puhdas ekstensionaalinen tieto. Intensionaalis-ekstensionaalisilla kyselyillä mahdollistetaan kyselyt, joissa vasta-

ukseksi saadaan molemman tason tietoa. Tämä kyselytyyppi on myös yleinen olemassaolevilla kyselykielillä.

Kielen pohjalla oleva mallinnustapa sitoo intensionaalisen ja ekstensionaalisen tason tavalla, joka mahdollistaa kaksisuuntaisen liikkumisen tasojen välillä. Tasojen sitomiseen ja osa-kokonaisuussuhteen analysointiin käytin indeksointimekanismia [Niemi, 1983; Junkkari, 2001]. Mallinnuksen toteutin Prolog++ ohjelmointikielellä [Moss, 1994], joka on deduktiivinen olio-orientoitunut kieli. Junkkarin [2001] kehittämä PSE-esitystavan mallinsin Prolog-termeillä konstruktointiprimitiivien avulla [Niemi and Järvelin, 1991].

Prototyypin kohdalla en kiinnittänyt suurta huomiota tehokkuusnäkökulmaan. Tietyt monimutkaiset kyselyt vievätkin aikaa. Jatkokehittelyn kannalta tähän asiaan tuleekin kiinnittää huomiota. Prototyyppi osoittaa kuitenkin mahdollisuuden kehittää kieli, joka on sekä erittäin deklaratiivinen että samalla ilmaisuvoimainen.

Totesin tutkielman teon edetessä, että myös kahdelle ekstensionaalis-ekstensionaaliselle primitiiville löytyy kielessä käyttöä. Intensionaalis-ekstensionaalisilla primitiiveillä ei pysty riittävän tehokkaasti sitomaan ilmentymiä toisiinsa tietyissä kyselytarpeissa. Ilman ekstensionaalis-ekstensionaalista primitiiviä ei pysty hakemaan tietyn olion komposiitti- tai komponenttioliota. Tällaiseen tilanteeseen on tarvetta esimerkiksi seuraavassa kyselyssä, joka liittyy edellä esitettyyn esimerkkitietokantaan: ”Anna sellaisen *bicycle*:n *price*-attribuutin arvo, jonka *frame*-komponentin *frame no*-attribuutin arvo on *8265*”. Tällaisessa kyselyssä tarvitaan primitiiviä, jolla pystytään löytämään tietyn olion komposiittiolio tai vaihtoehtoisesti tietyn olion komponenttiolio. Esitystapa mahdollistaa helposti ekstensionaalis-ekstensionaalisten primitiivien toteuttamisen ja tulevaisuudessa toteutetaankin tarvittavat kaksi primitiiviä.

Jatkossa kieli on tarkoitus integroida is-a suhteen monipuoliseen käsittelyyn soveltuvan kielen [Christensen, 1998; Niemi *et al.*, 2000] kanssa. Integroinnin myötä mahdollistuvat osa-kokonaisuussuhteiden hyväksikäyttö myös *deduktiivisten oliotyyppien* määrittelyssä. Deduktiivisten oliotyyppien ideana on ryhmitellä olioita siten, että kriteerit täyttävät oliot edustavat ilmeistä reaalimaailman käsitettä. Esimerkkinä deduktiivisesta oliotyypistä, joka määritellään osa-kokonaisuussuhteen pohjalta, voisi olla lasten polkupyörä. Kriteereinä tällaiselle polkupyörälle voisi olla se, että pyörä on joko kolmipyörä tai sen rungon korkeus on enintään 50 senttiä. Deduktiivista oliotyyppiä voidaan käyttää samalla tavalla kyselyissä kuin normaalia oliotyyppiä. Tavoitteena on, että loppukäyttäjän ei siis tarvitse tietää, onko kyseessä deduktiivinen oliotyyppi vai normaali oliotyyppi.

8. Lähdeluettelo

[Agrawal, 1987] Rakesh Agrawal, Alpha: An extension of relational algebra to express a class of recursive queries. In: *Proc.of the 3rd IEEE Conference on Data Engineering*, 580-590.

[Artale *et al.*, 1996] Alessandro Artale, Enrico Franconi and Nicola Guarino, Part-whole relations in object-centered systems: an overview. *Data & Knowledge Engineering*. **20** (1996), 347-383.

[Beraha and Su, 1999] Sabina Beraha and Jianwen Su, Support for modeling relationship in object-oriented databases. *Data & Knowledge Engineering*. **29** (1999), 227-257.

[Belford and Santone, 1989] G.G.Belford and A.L. Santone, Object-oriented database for construction data. *In: Proc. of the 22nd Hawaii International Conference on System Sciences.* (1989), 559-567.

[Christensen, 1998] Maria Christensen, Deduktiivisen oliotietokannan toteuttaminen ja siihen perustuvan kyselykielen kehittäminen loppukäyttäjälle. *Tampereen yliopisto,tietojenkäistteluyopin laitos,pro gradu tutkielma*,lokakuu 1998.

[Civello, 1993] Franco Civello, Roles for composite objects in object-oriented analysis and design. In: *Proc. of OOPSLA'93, ACM*. (1993) 376-393.

[Cluet, 1998] Sophie Cluet, Designing OQL: Allowing objects to be queried. *Information Systems* **23,**5 (1998), 279-305.

[Carey *et al.*, 1988] Michael J. Carey, David J. DeWitt and Scott Mohoric L. Vanderberg, A data model and query language for EXODUS In: *ACM SIGMOD International Conference on Management of Data* **17**, 5 (Sep. 1988), 413-423.

[Cattell *et al., 2000*] R.G.G Cattell, Douglas Barry, Mark Berler, Jeff Eastman, David Jordan, Craig Russell, Olaf Schadow, Torsten Stanienda and Fernando Velez, *The Object Data Standard: ODMG 3.0*, Morgan Kaufmann, 2000.

[Deux *et al.*, 1990] O. Deux and workgroup, The story of O_2. *IEEE Transactions on Knowledge and Data Engineering*. **2,** 1 (1990), 109-124.

[Gerstl and Pribbenow, 1995] Peter Gerstl and Simone Pribbenow. Midwinters, end games, and bodyparts: A classification of part-whole relations. *International Journal of Human-Computer Studies*, **43** (1995), 865-889.

[Goldstein *et al.*, 1999] Robert C. Goldstein, Veda C. Storey, Data abstractions: Why and how? *Data & Knowledge Engineering*. **29** (1999), 293-311.

[Halper *et al.*, 1998] Michael Halper, James Geller, Yehosua Perl, An OODB Part-Whole model: Semantics, notation and implementation. *Data & Knowledge Engineering*. **27** (1998), 59-95.

[Hanh *et al.*, 1999] Udo Hahn, Stefan Schulz, Martin Romacker. Part-Whole Reasoning: A Case Study in Medical Ontology Engineering. *IEEE Intelligent Systems.* **14**, 5 (1999) 59-67.

[Iris *et al.*, 1988] M. Iris, B. Lutowitz and M. Evens. Problems with the part-whole relation. In: M. Evens editor, *Relational models of the lexicon, (1988)* 261-288.

[Junkkari, 2001] Marko Junkkari, The systematic object-oriented representation for managing intensional and extensional aspects in modeling of part-of relationships. University of Tampere, Dept. of Computer and Information Sciences, Report **A-2001-5,** June 2001.

[Järvelin and Niemi, 1999] Kalervo Järvelin and Timo Niemi, Integration of complex objects and transitive relationships for information retrieval. *Information processing and management*. **35,** 5 (1999), 655-678.

[Kim *et al.*, 1987a] W.Kim, N.Ballou, J,Banerjee, H-T.Chou, J.F.Garza, and D.Woelk, Features of the ORION object-oriented database system In*: Proc. of the 13th international VLDB Conference. 319-329.*

[Kim *et al.*, 1987b] Won Kim, Jay Banerjee, Hong-Tai Chou, Jorge F. Garza and Darrell Woelk, Composite object support in an object-oriented database system. In: *Proc. of 2nd Conference on Object-Oriented Programming Systems, Languages, and Applications. 118-125.*

[Korth and Roth, 1987] Henry F. Korth, Mark A. Roth, Query language for nested relational databases. In*: Proc. of Workshop on Theory and Applications of Nested Relations and Complex Objects, Lecture Notes in Computer Science **361** (1987),* Springer-Verlag, 190-204.

[Koskimies, 2000] Kai Koskimies. *Oliokirja*, satku.fi, 2000.

[Lambrix, 2000] Patrick Lambrix, Part-Whole Reasoning in an Object-Centered Framework. *Lecture Notes in Artificial Intelligence* **1771** Springer, 2000.

[Lambrix and Padgham, 2000] Patrick Lambrix and Lin Padgham, Conceptual modeling in a document management environment using part-of reasoning in description logics. *Data & Knowledge Engineering* **32** (2000), 51-86.

[Lecluse *et al.*, 1988] Chritophe Lecluse, Philippe Richard and Fernando Velez, O2, an object-oriented data model. *ACM SIGMOD International Conference on Managing of Data* **17**, 5 *(Sep. 1988),* 424-433.

[Liu, 1999] Mengchi Liu, Deductive database languages: problems and solutions, *ACM Computing Surveys*, **31**, 1 (1999) 27-62.

[Moss, 1994] Chris Moss, Prolog++ The Power of Object-Oriented and Logic Programming. Addison-Wesley, 1994.

[Motro, 1994] Amihai Motro, Intensional answers to database queries. *IEEE Transaction on Knowledge and Data Engineering.* **6,**3 (Jun. 1994), 444-454.

[Motschnig-Pitrik and Kaasboll, 1999] Renate Motschnig-Pitrik and Jens Kaasboll, Part-whole relationship categories and their application in object-orinented analysis. *IEEE Transactions on Knowledge and Data Engineering*, **11,** 5 (1999), 779-796.

[Nahouraii and Petry, 1991] Ez Nahouraii and Fred Petry, *Object Oriented Databases,* IEEE Computer Society Press, 1991.

[Niemi, 1983] Timo Niemi, A seven-tuple representation for hierarchical data structures. *Information Systems.* **8**, 3, (1983), 151-157.

[Niemi *et al.*, 1998] Timo Niemi, Marko Junkkari and Kalervo Järvelin, Deductive object-oriented approach to systems analysis and its representation with the set theory. University of Tampere, Dept. of Computer and Information Sciences, Report **A-1998-15,** December 1998.

[Niemi *et al.*, 2000] Timo Niemi, Maria Christensen and Kalervo Järvelin, Query language approach based on the deductive object-oriented database paradigm. *Information and Software Technology **42** (2000), 777-792.*

[Niemi and Järvelin, 1991] Timo Niemi and Kalervo Järvelin, Prolog-based meta-rules for relational database representation and manipulation. *IEEE Transactions on Software Engineering.* **17**,8 (1991), 762-788.

[Niemi and Järvelin, 1996] Timo Niemi and Kalervo Järvelin, The processing strategy for the NF^2 relational frc-interface. *Information and Software Technology*. **38** (1996) 11-24.

[Paton *et al.*, 1996*]* Norman Paton, Richard Cooper, Howard Williams and Philip Trinder, *Database Programming Languages.* Prentice Hall, *1996.*

[Price *et al*., 2000] Modeling part-whole relationships for spatial data. In: *Proc. of the Eighth ACM Symposium on Advances in Geographic Information Systems*. 1-8.

[Rich and Knight, 1991] Elaine Rich, Kevin Knigth, *Artificial Intelligence*, McGraw-Hill, 1991.

[Roth *et al.*, 1988] Mark A. Roth, Henry F. Korth, and Abraham Silberschafz, Extended algebra and calculus for nested relational databases. *ACM Transactions on Database Systems*. **13**, 4 (1988), 389-417.

[Savnik *et al*., 1999] Iztok Savnik, Zahir Tari, Tomaz Mohoric. QAL: A query algebra of complex objects, *Data & Knowledge Engineering*, **30** (1999), 57-94.

[Sterling and Shapiro, 1986] Leon Sterling and Ehud Shapiro, *The Art of Prolog*, The MIT Press, 1986.

[Varzi, 1996] A.C.Varzi, Parts, wholes, and part-whole relations: The prospects of mereotopology, *Data & Knowledge Engineering* **20** (1996), 259-286.

[Wand *et al.*, 1999] Yair Wand, Veda C. Storey and Ron Weber, An ontological analysis of the relationship construct in conceptual modeling, *ACM Transactions on Database Systems* **24,** 4 (December 1999), 494-528.

[Winston *et al.*, 1987] Morton E. Winston, Roger Chaffin and Douglas Herrmann. A taxonomy of part-whole relations, *Cognitive Science,* **11** (1987), 417-444.